LIDAR Odometry with Joint Geometric and Appearance Landmarks

by

Benjamin Anargyros Peregrine Skikos

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Mechanical and Mechatronics Engineering

Waterloo, Ontario, Canada, 2019

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Odometry is the problem of estimating the motion of a moving platform relative to its environment without measurements to a fixed reference point. This is a critical problem for mobile robotics applications where measurements to a fixed reference point are not always assured, such as self-driving cars operating in GPS-denied environments and with physical landmarks potentially obscured by other traffic. Even when a fixed reference is available, motion estimates still improve position estimates by constraining the change in position over short timescales.

A fundamental limitation of odometry is that assuming some non-zero error, the path produced by integrating odometry will always diverge from the true path. The rate of divergence depends on the ego-motion and the environment. Aggressive motions such as rapid rotation or high acceleration are likely to cause greater error because they are more difficult to model compared to more sedate motions. Odometry that functions by comparing consecutive sensor samples to determine motion can exhibit greater drift due to high velocity because high velocity reduces the overlap between sensor samples. Lastly, unstructured portions of the scene may not contain useful information to fully constrain the ego-motion. A good example is moving next to a flat featureless wall since observations of that wall only constrain perpendicular motion.

In mobile robotics, both camera and lidar are commonly used for odometry. Lidar is a sensor technology that measures the time of flight of laser pulses to collect range-bearing samples from the scene. Lidars are used instead of cameras for certain applications despite their relatively high cost because lidars are not affected by ambient lighting conditions; they do not suffer from glare, or have to trade-off motion blur and sensitivity in low-light conditions. An ancillary benefit of using lidar is that the direct range measurement capability of lidar removes complexity from the odometry algorithm because the distance of sampled points does not have to be estimated from multiple sensor measurements.

Lidar odometry algorithms already exist yet there are opportunities for improvement. The intensity information collected by lidar commonly goes unused, with few of the top-performing lidar odometry algorithms on the Kitti odometry dataset leveraging it. Assuming that scenes lacking both geometric and appearance information are less likely than those lacking only geometric information, then a lidar odometry algorithm that leverages both types of information will be more robust than an algorithm that relies only on one type, all other things being equal. Robustness is desirable because it makes performance predictable.

The main contribution in this thesis is a lidar odometry algorithm that uses both appearance-based intensity landmarks as well as geometric landmarks to model the scene.

The intensity landmarks are gradient edges that model features in the environment such as lane markings while the geometric features are geometric edges and planes. Feature points are selected from multiple lidar scans and matched to the landmarks. The landmarks and the trajectory are jointly optimized within a sliding window filter. The addition of intensity landmarks improves performance in most cases compared to using geometric only landmarks.

Evaluation is carried out using the Kitti odometry metric on the Kitti odometry dataset as well as a dataset collected in Waterloo Ontario. Average drift on the Kitti training set was 4.6-9.6 deg/km and 0.71%-2.8% translation drift. Average drift on the Waterloo dataset was 3.1-3.4 deg/km and 1.1%-1.2% translation drift. Compared to a baseline configuration using only geometric landmarks, adding appearance-based landmarks produced a slight performance improvement on average across the Kitti subsequences as well as the Waterloo dataset, with up to a 13% reduction on translation error and a 6% reduction of rotation error for specific subsequences. However, there are also subsequences that exhibited worse performance with the addition of appearance-based features. In conclusion, the evaluation shows that the addition of appearance information can lead to better performance and that the method presented would benefit from additional work on further developing the scene appearance model and how it is applied.

Acknowledgements

I thank my advisor, Dr. Steven Waslander, for the incredible opportunity to work on this and other interesting projects (especially the moose) over the course of my degree as well as for providing invaluable guidance when I needed it.

I also thank the other members of the Wavelab that I have had the pleasure to work with. They are what made the Wavelab fun, exciting, productive, and comfortable. I wish the best to all of you.

Dedication

To my parents.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

When a robot is navigating through an unknown environment, a fundamental problem that must be solved is to track its own motion by creating a map of its environment. This is known as the simultaneous localization and mapping or SLAM problem. SLAM is a mature research area [22] [27] [39] with a number of solutions already in use in industry and is an integral part of many applications in robotics such as autonomous quad-copters and robotic vacuum cleaners [2]. Due to the endless combinations of robot, environment, and performance requirements, there are still unsolved problems in SLAM: most notably robustness in challenging environments [9].

One of the most exciting recent applications of SLAM is self-driving cars because they will provide far-reaching benefits once the technology is realized and reaches widespread adoption. 93% of all motor vehicle crashes in the US have driver error as the primary factor, and 31% of all fatal crashes involve alcohol. Removing purely human error will improve road safety [15], as well as removing the economic cost of those crashes.

A critical component of SLAM algorithms is the relative pose estimation between consecutive times at which sensor data is collected [38]. This is referred to as odometry, and it is one of the core components of any SLAM system. SLAM systems combine odometry with other techniques in order to produce a trajectory that is with respect to a single inertial frame of reference.

An essential technique used in SLAM is loop closure. A loop is any part of the trajectory that leaves from somewhere and then revisits the same spot. SLAM algorithms use place recognition to recognize when the robot has traveled back to the start of a loop, and adjusts the trajectory estimate over the loop to ensure that the estimated path starts and ends in the same spot. This compensates for error that has accumulated over the loop.

Robust odometry is important for robust SLAM because having predictable odometry error reduces the problem size for place recognition; only previously visited places that are within a reasonable search area from the current position estimate need be considered. If the odometry is not robust, the odometry error is not reliable and a larger search area is required. Furthermore, if the trajectory does not contain any loops, than the SLAM trajectory is highly reliant on the integrated odometry and robust odometry translates directly to robust SLAM.

Odometry is evaluated by comparing the path produced by integrating relative pose estimates against the true path. As each relative pose estimate contains some error, the integrated path estimate diverges from the true path as the length of the path increases. Odometry error is therefore evaluated in terms of error per distance traveled in order to take into account the effect the length of the path has on the error. An odometry algorithm is characterized by comparing the rate of divergence under any operating conditions of interest. A perfectly robust odometry algorithm would have the same rate of divergence in all operating conditions.

Modern odometry used with SLAM relies upon sensors that sample the scene, then use those measurements to track the scene's movement. Broadly, information about the scene can be categorized as geometric or appearance-based. Measurements of how surfaces are distributed in the scene contain geometric information while measurements of the intensity/colour of surfaces in the scene contain appearance information. Both of these types of information can be used separately for odometry, but using both leads to a more robust odometry algorithm because if a scene lacks one type of information it still might contain the remaining type.

The use of cameras in odometry is well-studied [9] [20] because cameras capture rich measurements of the scene while being inexpensive. Odometry based on vision is classified as visual odometry (VO). Cameras collect rich appearance information from a scene, making it possible to recognize features in the scene when they appear in multiple pictures. By tracking features through multiple frames, the motion of the camera as well as the feature locations can be recovered up to a scale ambiguity.

Scale ambiguity is a characteristic of monocular VO and it comes about because cameras do not measure the depth of whatever they are imaging. This limitation means that the images of an object and of a one-quarter scale copy of the object placed four times closer to the camera are identical. Consequently, the scale of both the map and trajectory produced by monocular VO systems is unobservable.

The problem of absolute scale recovery in VO can be solved by combining inertial measurement units (IMUs) with VO to make visual-inertial odometry (VIO). IMU mea-

surements are complementary with vision because they can measure short term changes in pose, which allows the recovery of scale in VIO systems. Methods that use multiple cameras are also capable of recovering scale when the poses of the cameras with respect to one another are known. More recently, work on VO incorporating machine learning techniques [37] [11] has demonstrated scale recovery [37] with a monocular system, however the overall performance is less than state-of-the-art VO that does not use machine learning.

The largest limitation of a camera is that they are reliant on ambient light. Cameras sample a scene by focusing an image of the scene onto photosensitive elements arranged in a grid, with each element roughly corresponding to a pixel in the photo. These elements integrate the intensity of impinging light during an exposure period to produce an intensity measurement. The elements have both a minimum detection threshold as well as a saturation threshold. The dynamic range of the camera is the ratio between the maximum and minimum intensities the camera can measure.

If a scene's lighting range exceeds the dynamic range of the camera then details in the scene will be lost in pixels that are below the detection threshold or above the saturation threshold. These conditions are referred to as under- and overexposed respectively. Additionally, although using a long exposure time in dark environments can avoid underexposure, a long exposure time is not practical in dynamic environments because any movement during the exposure period creates motion blur in the image. Blurring reduces image sharpness, and can cause VO to lose track of features.

Limited dynamic range being an issue is not an uncommon occurrence. From the perspective of self-driving cars, operating at night on an unlit road results in the scene having a wide range of intensities between oncoming headlights and the ground. As in this example it is not always possible to adjust the intensity range of the scene, and so any VO in this application will be degraded by a loss of detail in the images.

A lidar is a type of sensor named for its operating principle of light detection and ranging (LIDAR). Pulses of light, typically from a laser, are emitted and the time taken for the reflection to return to the sensor as well as the intensity of the returned pulse is recorded. It is a more costly sensor than a camera, but despite all the work done on VO, the active illumination of LIDAR remains a fundamental advantage compared to vision because it removes the dependency on ambient lighting conditions. The direct range measurements from LIDAR also result in a lack of scale ambiguity for odometry based on a single lidar.

LIDAR has its own unique characteristics that must be taken into consideration when designing a LIDAR odometry (LO) algorithm. The lidars used in mobile robotics have anisotropic resolution, the intensity measurements from LIDAR are not viewpoint invariant, and the sensor samples the scene continously instead of in short exposure periods.

Lidars used for robotic applications have coarser resolution than cameras[1] and so are unable to resolve fine detail to the same extent. In addition, the vertical resolution is coarser than the horizontal resolution, which makes applying traditional vision-based appearance methods challenging, as they assume isotropic resolution.

The intensity of the returned laser pulse varies with the viewpoint because it is affected by the range of the point, the angle of incidence of the laser to the surface, the reflectivity of the surface, and the specularity of the surface. These effects must be compensated for if viewpoint-invariant intensity measurements are required. This increases algorithmic complexity because compensating for these effects requires surface normal estimation to find the angle of incidence and the coarse resolution of lidar in turn makes surface normal estimation more difficult.

The exposure period of cameras is typically brief and approximating it as a single instant in time in order to use a discrete-time state formulation is a common simplification for VO. In contrast, rotating lidars like the Velodyne HDL-64e sample continuously from the environment. Due to the high frequency at which a lidar samples points, using a discrete-time state formulation is not feasible, so a non-trivial continuous-time formulation is required.

The ability to provide precise depth measurements in any lighting conditions can potentially outweigh the challenges of continuous sampling and coarser resolution, motivating the use of LIDAR odometry (LO) despite the higher sensor cost. This work focuses on extending an existing state-of-the-art LO algorithm to use appearance information from lidar intensity, optimize over multiple lidar scans simultaneously, and incorporate the continuous-time framework of another LO method, which is a suitable solution for the continuous-time nature of the lidar used.

1.1 Related Works

Scan Registration

LO is similar to scan registration and algorithms used for scan registration can be adapted for odometry. However, the goal of scan registration is to position one or more overlapping

[1]The number of pixels measured by a single camera in the Kitti dataset is about 4.5 million pixels per second distributed over about a 70 degree field of view while the lidar (Velodyne HDL-64e) measures about 1.33 million points per second distributed over about a 360 degree horizontal field of view and a 27 degree vertical field of view

scans to form a consistent overall picture of the environment while the goal of odometry is to recover the ego-motion of the sensor as it travels through its environment. In general, the scan registration problem does not include knowledge about the kinematics/dynamics of the sensor or distortion in the scans.

Scan matching is a rich area in its own right [34]. Some challenging areas for scan matching include convergence from poor initialization, convergence in geometrically ambigious areas, and convergence in a dynamic scene. In LO, the scans are processed sequentially and a good initial estimate for the trajectory is available from the motion model. This removes the need for a wide convergence capability, simplifying the problem.

The contribution from scan matching used in this work is that the proposed algorithm performs iterative closest point data association similiar to the iterative closest point family of scan registration algorithms [34].

ICP Family of Scan Registration

This class of scan matching algorithms work by finding corresponding points between scans and then minimizing some measure of error given the correspondences by adjusting the relative position of the scans. These steps are done iteratively until a stopping criteria is met, such as the solution changing less than a threshold. The class is named after one of the first (and simplest) such algorithms, Iterative-Closest-Point, introduced by [6].

Given two sets of points $P = \{p_1, p_2, ..., p_n\}$ and $Q = \{q_1, q_2, ..., q_m\}$ where $p_i, q_i \in \mathbb{R}^3$, there is assumed to be correspondences $C = \{\{p_x, q_x\}_1, ..., \{p_y, q_y\}_k\}$ such the points in each pair are the same point in the scene when expressed in a common frame. Under this assumption, there exists a transform $T \in SE(3)$ that maps each point in P to its corresponding point in Q. Given correspondences, error is formulated with respect to the transform

$$c_{ICP}(T) = \sum_{i=0}^{k} \|Tp_i - q_i\|^2 \tag{1.1}$$

where i indexes each pair of correspondences. The optimal transform is found by minimizing the error. Finding the set of correspondences is done by transforming P with the best estimate of T, then finding the nearest Euclidean neighbour in Q.

The assumption that there are points in each pointcloud corresponding to the same physical location is problematic because it is unlikely that a lidar will sample the same

locations after it has moved. For lidars with fine resolution, such as those used in surveying, it is a reasonable simplification, however lidars used for odometry in mobile robotics typically trade-off resolution in order to increase scan frequency. For these applications the correspondence assumption can be a source of significant systematic error.

This limitation has lead to variants of ICP that relax the perfect correspondence assumption. Most variants incorporate some prior knowledge about the structure of the scene to improve convergence, increase robustness, and improve accuracy. A few examples of ICP variants are Generalized-ICP and normal distributions transform matching. Generalized-ICP [31] assumes that there are surfaces in the scan that are locally planar, and models each point as being sampled from a Gaussian distribution. The shape of the Gaussian distribution is set such that there is high variance along the plane and low variance along the plane normal. The ICP cost function is modified to weigh the error for each correspondence according to those distributions. Normal distributions transform matching [7] converts the pointclouds into a number of Gaussian distributions, and then minimizes the distribution-to-distribution distance to find the optimal transform.

ICP has also been extended to optimize over a non-rigid transforms. In [24], additional local transformation parameters were introduced for each of the different sample times throughout the scans, and the cost function modified to penalize the difference between temporally-adjacent transformation parameters to provide smoothing. This produces smooth variation across time, but induces systematic error because it does not smooth in a manner consistent with the kinematics/dynamics of the ego-motion. It also greatly increases the dimensionality of the problem.

LOAM

Lidar Odometry And Mapping is one of the state-of-the-art LO algorithms [40]. The main contribution is to match points from scan to scan based on local geometry, using distance functions that reflect the underlying geometry being sampled by the Lidar, and that the formulation is simple enough that it can be run in real-time. The points measured by the lidar are used to implicitly define all the geometry in the scene. It was developed with anisotropic and 1D sensors in mind.

Distortion due to ego-motion is handled in a hierarchical manner: first, measurements from an IMU are used to undistort the component of distortion caused by non-zero acceleration that occurred during the scan. It is assumed any remaining distortion will be linear; caused by bias in the IMU or error in the initial velocity used with the IMU. A

transform parameterization that may be linearly interpolated over the scan is optimized to correct for any remaining linear distortion.

The contribution of LOAM used in this work is the method of extracting feature points from sparse lidar scans.

Simultaneous Trajectory Estimation and Mapping

STEAM is a continuous-time SLAM formulation. These formulations are uncommon compared to discrete-time SLAM because commonly-used sensors (such as cameras with global shutters) do not spread their measurements evenly across time, and IMU measurements can be pre-integrated to incorporate high-frequency IMUs with low-frequency states [17].

STEAM uses Gaussian Process interpolation as the basis for its formulation, but their also exist continuous-time formulations that use temporal basis functions such as Kalibr[19], which is a calibration toolbox able to find time delays between multiple sensors.

The contributions of the STEAM formulation used in this work is the ability to interpolate for each measurement time from a small set of states in constant time, and a general motion model.

Surfels

Surfel is a portmanteau of surface element, and surfels have been used with some success for mapping in lidar. The scene is divided, by discretization or other techniques, into small volumes that are modeled as surfels [8] [14].

A variety of surfel models have been used, ranging from ellipsoidal to cylindrical. Surfels can be implicitly defined by measurement points, for example the mean and covariance of a set of points, that do not add any additional optimization parameters, or may have explicit parameterizations that add degrees of freedom to the problem.

Surfels is used in this work to model the scene.

Lidar intensity channel

Lidars return the intensity of each returning laser pulse, so surfaces that reflect more are distinguishable. Lidar intensity information has been used with 2.5D grid maps for surface

vehicles[26]. Each cell contains a probabilistic model for the intensity of returns falling in that cell, and by maximizing the likelihood of all the returns by adjusting the cell models and scan positions, the map and trajectory is recovered.

Intensity has also been combined with more general scan registration techniques, for example by extending ICP-type algorithms to incorporate it as an additional channel/dimension [32]. Intensity features are modeled as surfels in this work to improve the overall scene model.

1.2 Outline and Contributions

The contributions made in this work are to:

- Formulate LIDAR odometry using explicit surfel landmarks within a sliding window.
- Incorporate the intensity information from LIDAR using surfels.

Sliding-window filters have improved performance compared to single-step filters, yet still have bounded runtime. LOAM does not scale well to a sliding window filter because it operates on a fairly large number of low-level feature points. The number of active feature points grows linearly with the number of scans, and can make a sliding window filter impractical. Using explicitly-parameterized surfels as landmarks and matching between landmarks and feature points rather than directly between feature points reduces the complexity of the problem because the distribution and number of landmarks can be limited according to a sampling strategy, and only feature points near landmarks need be considered when matching, reducing the size of the set that must be searched for correspondences.

A sliding-window filter also allows the landmark parameters to be optimized jointly with the trajectory, and the incorporation of a robust loss function on landmark error to reduce the impact of noisy feature points in any single scan.

The use of intensity features as well as geometric features adds robustness for scenes that are lacking in one or the other. In this work, the feature extraction method from LOAM is modified to locate intensity gradients, which are then fit to edge landmarks. Using intensity gradients instead of intensity directly avoids the previously introduced systematic errors in lidar intensity from impacting the estimation because lidar intensity error tends to preserve gradients. Edge landmarks are used as these are reasonable models

for linear lane markings and boundaries between dissimilar materials on walls, such as windows or doors.

The remainder of this thesis is broken into background, the formulation of the proposed method, its evaluation, and discussion. Chapter 2 introduces any existing concepts, conventions, or algorithms used in this work. Chapter 3 presents the design of the proposed method as well as some implementation-specific details. Chapter 4 presents results obtained by the proposed method on the Kitti dataset [20] and on data collected in Waterloo, ON. Finally, Chapter 5 explores the results obtained and future work that could provide improvement.

Chapter 2

Background

2.1 Transforms and Coordinate Systems

Due to the wide range of transform conventions used by various groups, it is necessary to explain the convention used in this work. A transform is defined as the operation that takes a 3D point expressed in a Cartesian coordinate system and expresses it in another Cartesian system, where the two systems differ by a translation and a rotation. All coordinate systems are right-handed. This type of transform is called a rigid transform and belongs to the SE(3) group [5], a group being a set having a group operation, an inverse operation, and an identity element. The importance of transforms being part of this group will be made more apparent in Section 2.2. The transform parameterization is

$$T \in SE(3) = \{T|T = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix}, R \in \mathbb{R}^{3x3}, t \in \mathbb{R}^3, R^T R = I, \det R = 1\} \tag{2.1}$$

where R parameterizes the rotation and t parameterizes the translation

Within this thesis, transforming a point from coordinate system A to coordinate system B refers to a point with respect to A being transformed such that it is with respect to B given the rigid transform between A and B. In that example, the transform would be denoted at T_{BA}. For clarity, points are also subscripted with the frame they are expressed in. For example, point P expressed in frame A would be written as ${}_AP$. Finally, vector or physical quantities such as velocity have three pieces of information subscripted to fully describe the reference frames involved. For example, the velocity of frame A with respect to frame B expressed in frame C is written as ${}_Cv_{BA}$. This follows the convention introduced by [18].

For SE(3), the group operation is transform composition

$$T_{02} = T_{01}T_{12} = (R_{02} = R_{01}R_{12}, {}_0t_{02} = R_{011}t_{12} + {}_0t_{01}) \tag{2.2}$$

where $T_{01}, T_{12}, T_{02} \in SE(3)$. As the operation is defined for any two elements in SE(3), subscripts are used to ensure that the operation is only carried out for compositions that make physical sense: composing the transform from frame 2 to frame 1 with that from frame 1 to frame 0 gives the transform from frame 2 to frame 0.

The group inverse operation

$$\begin{aligned} T_{10} &= T_{01}^{-1} \\ T_{01}^{-1} &= \begin{bmatrix} R_{01}^T & -R_{010}^T t_{01} \\ 0 & 1 \end{bmatrix} \end{aligned} \tag{2.3}$$

where $T_{01} \in$ SE(3) has the physical meaning of reversing the transformation direction. The identity element is the 4x4 identity matrix.

Transforming points between frames

$$\begin{aligned} {}_Bp &= T_{BA}{}_Ap \\ {}_Bp &= R_{BA}{}_Ap + {}_Bt_{BA} \end{aligned} \tag{2.4}$$

where $T_{BA} \in$ SE(3), ${}_Ap \in \mathbb{R}^3$ changes the frame the point is expressed in. This function is defined for any combination of SE(3) element and $\mathbb{R}^3$ element, so as with the transform composition operation, subscripts are used to prevent invalid point transformations. In this case, the subscript for the frame the transform is transforming from must match the subscript for the frame of the point.

The subscripts are used in this work whenever composition or transformation is taking place, otherwise they are omitted to simplify equations. A pose refers to how one coordinate system is positioned with respect to another. The pose of A with respect to B is equivalent to T_{BA}. In this work, a trajectory is parameterized by a finite set of poses with associated body centered velocity. Twist elements, $\varpi \in \mathbb{R}^6$, parameterize linear and angular velocity.

2.2 Optimization on SE(3)

The set of all rigid transforms is the SE(3) Lie group. This is important from a state estimation perspective because transforms being a Lie group has implications on how

transforms are optimized. Details about Lie groups will only be introduced in this work as necessary for the proposed algorithm, but readers interested in a more complete background are directed to [5] for information on the use of Lie groups in pose estimation and [25] for information on Lie groups from a general topology perspective.

The difficultly with optimizing on SE(3) directly is that there does not exist a minimal parameterization for SE(3) that can uniquely represent the entire group. Over-parameterized representations for SE(3) uniquely cover the group, but contain constraints. As unconstrained optimization is easier then constrained optimization, the typical solution for optimizing poses is to optimize a minimally parameterized perturbation rather than the pose itself. Provided the perturbation is kept small, the region of SE(3) not uniquely covered can be avoided.

In this work, the minimal parameterization for the perturbation is the Lie algebra of SE(3). The Lie algebra is the tangent space of the group's identity element and is denoted as se(3). Elements of $\mathbb{R}^6$ can be mapped to elements in se(3) because $\mathbb{R}^6$ is isomorphic to se(3). This mapping

$$\begin{aligned} &\wedge : \mathbb{R}^6 \to \mathbb{R}^{4x4} \\ &x^\wedge := \begin{bmatrix} 0 & -x_3 & x_2 & x_4 \\ x_3 & 0 & -x_1 & x_5 \\ -x_2 & x_1 & 0 & x_6 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{aligned} \tag{2.5}$$

is given the $\wedge$ operator and is called the lift map. The $\vee$ operator is given to the inverse lift map.

For matrix Lie groups, elements of its Lie algebra can be mapped to elements in the Lie group through the matrix exponential. The matrix exponential restricted to the domain of a specific Lie algebra is called the exponential map for that Lie group. Due to the special structure of se(3), there is a closed-form expression for the SE(3) exponential map, even though a general closed-form matrix exponential expression does not exist.

The inverse of the exponential map is the log map, which takes elements of SE(3) to se(3) and also has a closed form expression due to the special structure of the group. Note that the exponential map is surjective and not injective: there are multiple elements of se(3) that can map to the same element of SE(3).

A characteristic of Lie groups is that each one has a group operation that takes two elements of the group and produces another element within the group. For SE(3), the group operation is transform composition. Therefore, after using the lift and exponential

maps to transform an element of $\mathbb{R}^6$ into an element of SE(3), transform composition can be used with another element of SE(3) to produce a new element of SE(3). Done in sequence, this process can be used to define a pseudo-addition operation

$$T \boxplus \epsilon := \exp(\epsilon^\wedge) T \tag{2.6}$$

where $\boxplus : SE(3) \times \mathbb{R}^6 \to SE(3)$ is the operator given to the combination of lift map, exponential map, and transform composition.

When using nonlinear optimization to find an optimal element of SE(3), the problem is reformulated as finding the optimal se(3) perturbation to the element of SE(3). Then once a perturbation is found, it is applied to the SE(3) element using the $\boxplus$ operator before moving to the next iteration of optimization. This is similar to regular nonlinear optimization, except that the optimal update is not simply added to the linearization point, instead $\boxplus$ is used.

A complication of using this optimization scheme is that $\boxplus$ is nonlinear with respect to the perturbation. This is not a problem so long as the Jacobian of $\boxplus$ is accounted for in the optimization, but defining a Jacobian requires a way to express the distance between two elements of SE(3). The distance is defined

$$T_i \boxminus T_j := \ln(T_i T_j^{-1})^\vee \tag{2.7}$$

where $\boxminus : SE(3) \times SE(3) \to \mathbb{R}^6$ is the pseudo-subtraction operator. This definition is such that $(T \boxplus \epsilon) \boxminus T = \epsilon, T \in \mathrm{SE}(3), \epsilon \in \mathbb{R}^6$.

Using $\boxminus$, the Jacobian of the exponential map

$$J(\xi) = \lim_{\epsilon \to 0} \frac{\exp((\xi + \epsilon)^\wedge) \boxminus \exp(\xi^\wedge)}{\epsilon} \tag{2.8}$$

can be defined. A closed form exists, and there are fast approximations when ξ is small [5]. Particularly, if ξ is zero than the Jacobian is identity. Having the Jacobian available is important because common nonlinear optimization techniques such as Levenburg-Marquadt require the Jacobian of the objective function.

Beyond this point, wherever a Jacobian is said to be with respect to a transformation, its meaning is that the Jacobian is with respect to a perturbation composed with that transformation

$$T_{new,i} = T_i \boxplus \epsilon_i^\wedge \tag{2.9}$$

rather than the transformation itself.

2.3 Gaussian Processes and Gaussian Process Interpolation

Given a multivariate Gaussian distribution, assume each dimension corresponds to the output of some time-varying function at a particular time. Sampling that distribution produces a sample of a discrete time series. A Gaussian process defines how to produce a multivariate Gaussian distribution given a set of indices. In this example, the index variable is time, but it can also be spatial. More formally, a Gaussian process is a stochastic process with the property that the set of outputs of the process given any finite collection of indices is normally distributed.

Gaussian processes (GPs) will be written as

$$y = GP \sim (m(t), \kappa(t, t')) \tag{2.10}$$

where t as an index variable, $m(t)$ is a mean function, and $\kappa(t, t')$ is a covariance kernel. The kernel defines the covariance between two sample points given their indices. The choice of mean function and kernel are application-specific, and are selected with some prior knowledge about the most likely functions. For example, a common criteria is that rapidly changing functions should have a low likelihood in order to dampen high-frequency noise. Given a set of sample points y with indices t, a Gaussian process defines a multivariate Gaussian distribution

$$\begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \sim N \left\{ \begin{bmatrix} m(t_1) \\ \vdots \\ m(t_n) \end{bmatrix}, \quad \begin{bmatrix} \kappa(t_1, t_1) & \dots & \kappa(t_1, t_n) \\ \vdots & \ddots & \vdots \\ \kappa(t_n, t_1) & \dots & \kappa(t_n, t_n) \end{bmatrix} \right\} \tag{2.11}$$

where $t = [t_1, ..., t_n]$ is the set of indices.

Given two sets of times, t_1, t_2 the distribution can be formed as

$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} \sim N \begin{pmatrix} m(t_1) \\ m(t_2), \end{pmatrix} \begin{bmatrix} K(t_1, t_1), K(t_1, t_2) \\ K(t_2, t_1), K(t_2, t_2) \end{bmatrix} \tag{2.12}$$

where $K(t_i, t_j)$ is the permutation matrix generated from the kernel function for each combination of sets t_i, t_j.

The posterior distribution of y_1 conditioned on $y_2 = Y_2$

$$p(y_1 | y_2 = Y_2) = \bar{y_1} \sim N(\bar{\mu}, \bar{\Sigma}) \tag{2.13}$$

where Y_2 are observations of y_2 is available through marginalization.

$$\begin{aligned} \bar{\mu} &= m(t_1) + K(t_1, t_2)K(t_2, t_2)^{-1}(Y_2 - m(t_2)) \\ \bar{\Sigma} &= K(t_1, t_1) - K(t_1, t_2)K(t_2, t_2)^{-1}K(t_2, t_1) \end{aligned} \tag{2.14}$$

Since the times in t_1 can be arbitrarily chosen, this technique can be used to interpolate between adjacent elements in t_2, thereby allowing a continuous trajectory to be parameterized with a finite number of discrete states.

2.4 Motion Model

A summary of the constant velocity motion model from [1] is presented. To start,

$$\begin{aligned} T(t) &\in SE(3) \\ \varpi(t) &\in \mathbb{R}^6 \end{aligned} \tag{2.15}$$

where $T(t)$ is the pose and $\varpi(t)$ is the body-centered velocity define the trajectory parameterization. The motion model

$$\begin{aligned} \dot{T}(t) &= \varpi(t)^{\wedge} T(t) \\ \dot{\varpi}(t) &= \omega'(t) \end{aligned} \tag{2.16}$$

is a constant velocity model subject to noise on acceleration where $\omega'(t)$ is the noise. The noise is a Gaussian process

$$\omega'(t) \sim GP(0, Q'_C \delta(t - t')) \tag{2.17}$$

where Q'_C is a covariance matrix and $\delta(t - t')$ is the Dirac delta function. This choice of mean and kernel functions specifies additive white noise. The covariance itself is a tuning parameter that adjusts the weight given to the constant velocity model in the optimization.

In order to make use of techniques developed for linear Gaussian processes, a local process model

$$\begin{aligned} T_k &= T(t_k) \\ \xi_k(t) &= T(t) \boxminus T_k \\ \dot{\xi}_k(t) &= J(\xi_k(t))^{-1} \varpi(t) \\ \ddot{\xi}_k(t) &= \omega(t) \\ w(t) &\sim GP(0, Q_C \delta(t - t')) \end{aligned} \tag{2.18}$$

where $J(\xi_k(t))^{-1}$ is the Jacobian of the log map and $t_k \leq t \leq t_{k+1}$, $\forall t$ is used. The Jacobian of the log map can be found by inverting the result of Eq. 2.8, or through a different closed-form expression. This model defines a local process between each pair of discrete states being optimized. The local model is useful because $J(\xi_k(t))^{-1}$ can be reasonably approximated as identity when there is a small amount of noise and/or $\xi_k(t)$ is small (the spacing between states can be adjusted to help this assumption). With this simplification the system is linear and GP marginalization can be used to interpolate states.

The prior on each local process is defined by integrating a linear, time-varying stochastic differential equation (LTV-SDE)

$$\begin{aligned} \dot{x}(t) &= F(t)x(t) + v(t) + L(t)w(t) \\ w(t) &\sim GP(0, Q_C\delta(t-t')) \end{aligned} \tag{2.19}$$

where $x(t)$ is the state, $F(t)$ captures a dependence of the derivative of the state on the current state, $v(t)$ represents an input, $w(t)$ is the noise, and $L(t)$ describes the relation between the noise to the derivative of the state.

The solution to an LTV-SDE [4] is

$$x(t) = \Phi(t, t_0)x(t_0) + \int_{t_0}^{t} \Phi(t, s)(v(s) + L(s)w(s))ds \tag{2.20}$$

where $\Phi(t, t_0)$ is known as the transition matrix or transition function.

The transition matrix must satisfy

$$\begin{aligned} \Phi(t, t) &= I \\ \dot{\Phi}(t, s) &= F(t)\Phi(t, s) \\ \Phi(t, s) &= \Phi(t, r)\Phi(r, s) \end{aligned} \tag{2.21}$$

but a general formula for the transition matrix given the LTV-SDE [5] does not exist.

Equation 2.20 is a Gaussian process, and expressions for the mean and kernel functions can be found by taking the expectation $\mu(t) = E[x(t)]$

$$\mu(t) = \Phi(t, t_0)x_0 + \int_{t_0}^{t} \Phi(t, s)v(s)ds \tag{2.22}$$

and by using the definition of the covariance $\kappa(t, t') = E[(x(t) - \mu(t))(x(t') - \mu(t'))^T]$

$$\kappa(t, t') = \Phi(t, t_0)K_0\Phi(t', t_0)^T + \int_{t_0}^{(t,t')} (\Phi(t, s)L(s)QL(s)^T\Phi(t', s)^T ds) \tag{2.23}$$

leading to

$$x(t) \sim GP(\mu(t), \kappa(t, t')) \tag{2.24}$$

as the LTV-SDE solution expressed as a GP [5].

With the GP defined, Eq. 2.13 can now be used to interpolate the state at any time t_τ of interest. For Eq. 2.19, because the noise is not correlated across time, the state at time t_τ depends only on the previous and next states. Given estimates of the states at times t_k and t_{k+1}, the posterior distribution of the state at time t_τ conditioned on the estimate of the states

$$p(x(t_\tau)|x_{t_k} = X_k, x_{k+1} = X_{k+1}) \sim N(\bar{\mu}(t_\tau), \bar{\Sigma}(t_\tau)) \tag{2.25}$$

where $t_k \leq t_\tau \leq t_{k+1}$ is found through marginalization. The conditioned mean

$$\bar{\mu}(t_\tau) = \mu(t_\tau) + \begin{bmatrix}\Lambda(t_\tau) & \Psi(t_\tau)\end{bmatrix} \left\{ \begin{bmatrix} X_k \\ X_{k+1} \end{bmatrix} - \begin{bmatrix} \mu(t_k) \\ \mu(t_{k+1}) \end{bmatrix} \right\} \tag{2.26}$$

and conditioned covariance

$$\Sigma = \kappa(t_\tau, t_\tau) + \begin{bmatrix}\Lambda(t_\tau) & \Psi(t_\tau)\end{bmatrix} \left\{ \begin{bmatrix} \Sigma_{k,k} & \Sigma_{k,k+1} \\ \Sigma_{k+1,k} & \Sigma_{k+1,k+1} \end{bmatrix} - \begin{bmatrix} \kappa(t_k, t_k) & \kappa_(t_k, t_{k+1}) \\ \kappa_(t_{k+1}, t_k) & \kappa_(t_{k+1}, t_{k+1}) \end{bmatrix} \right\} \begin{bmatrix}\Lambda(t_\tau)^T \\ \Psi(t_\tau)^T\end{bmatrix} \tag{2.27}$$

are expressed sing $\Psi(t)$, $\Lambda(t)$, and Q_t for compactness. Those terms are defined by

$$\begin{aligned} \Psi(t_\tau) &= Q_\tau \Phi(t_{k+1}, t_\tau)^T Q_{k+1}^{-1} \\ \Lambda(t_\tau) &= \Phi(t_\tau, t_k) - \Psi(t_\tau)\Phi(t_{k+1}, t_k) \\ Q_t &= \int_{t_k}^{t} \Phi(t, s) L(s) Q_C L(s)^T \Phi(t, s)^T ds \end{aligned} \tag{2.28}$$

and result from applying marginalization after substituting the expressions for the mean and kernel functions.

Expressing the system in Eq. 2.18 (approximating the inverse left Jacobian of SE(3) as identity) as

$$\begin{bmatrix} \dot{\xi}_k(t) \\ \ddot{\xi}_k(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \xi_k(t) \\ \dot{\xi}_k(t) \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} w(t) \tag{2.29}$$

exposes $F(t)$ and $L(t)$ for this specific system. This system also has a closed-form transition matrix

$$\Phi(t_{k+1}, t_k) = \begin{bmatrix} 1 & t_{k+1} - t_k \\ 0 & 1 \end{bmatrix} \tag{2.30}$$

which meets all the constraints for a transition matrix.

Substituting the specific $F(t)$, $L(t)$ and $\Phi(t', t)$ for this system into the general solution for the interpolated mean and covariance for an LTV-SDE allows the pose to be interpolated continuously along a trajectory parameterized by a finite set of discrete states.

Aside from interpolation, the motion model is also used to produce constraints between consecutive discrete states. The model in the local state variables

$$\begin{bmatrix} \xi_k(t_{k+1}) \\ \dot{\xi}_k(t_{k+1}) \end{bmatrix} = \Phi(t_{k+1}, t_k) \begin{bmatrix} \xi_k(t_k) \\ \dot{\xi}_k(t_k) \end{bmatrix} \tag{2.31}$$

is converted back to the global state to express each error. The error between the states at times t_k, t_{k+1}

$$e_m(x_k, x_{k+1}) = \begin{bmatrix} T_{k+1} \boxminus T_k - (t_{k+1} - t_k)\varpi_k \\ J(T_{k+1} \boxminus T_k)^{-1}\varpi_{k+1} - \varpi_k \end{bmatrix} \tag{2.32}$$

where $x_k = \begin{bmatrix} T_k \\ \varpi_k \end{bmatrix}$ has associated uncertainty $\Sigma_{e_m(x_k, x_{k+1})} = Q_{t_{k+1}}$. The total motion error

$$\mathcal{E}_m = \sum_{i=1}^{N-1} E_m(x_k, x_{k+1}) \tag{2.33}$$

where N is the number of discrete states and

$$E_m(x_k, x_{k+1}) = e_m(x_k, x_{k+1})^T \Sigma_{e_m(x_k, x_{k+1})} e_m(x_k, x_{k+1}) \tag{2.34}$$

is the sum of the weighted squared errors for each pair of states. This is added to the feature errors and jointly minimized.

2.5 Lidar Sensor

The type of lidar sensors used for automotive applications have a distinct anisotropic sampling pattern that can make the application of general scan matching algorithms challenging. Figure 2.1 is an example of a scan collected from a lidar. It shows how the

laser-detector pairs in this type of sensor produce distinct rings. The azimuthal resolution of this sensor depends on the rotation rate of the sensor and is much denser than the elevation resolution which is dependent on the number of laser-detector pairs and how they are mounted.

As well as showing the distribution of points collected by lidar, the Figure also provides a qualitative sense for the quality of the data. The geometric aspects of the scene can be clearly made out, especially in the areas having dense elevation resolution. In addition, the utility of the appearance information is also visible, with the doors & windows easily identifiable in the scan.

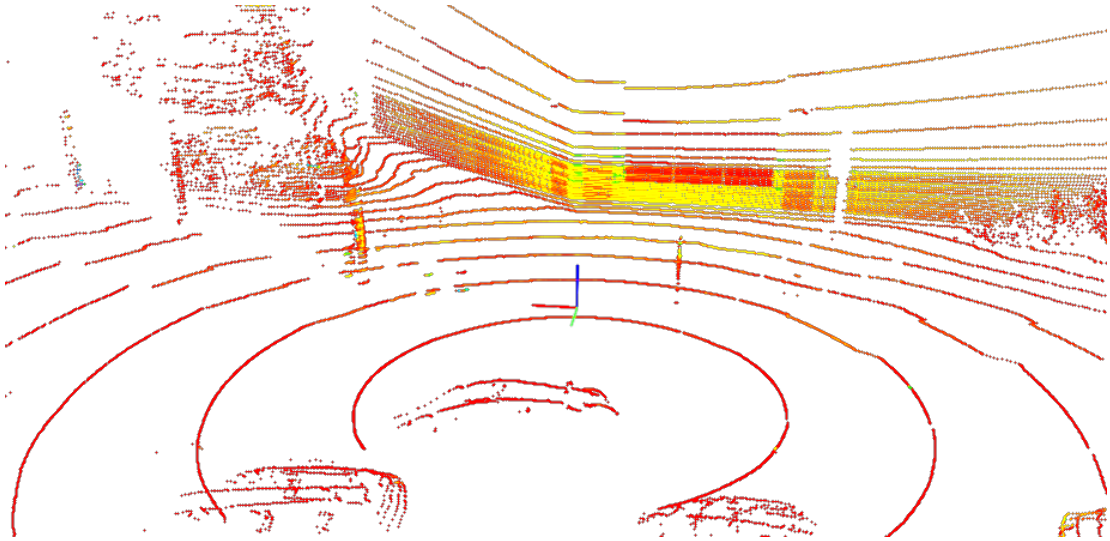

Figure 2.1: A scan from a Velodyne VLP-32C lidar mounted on top of a car waiting at an intersection. Points are coloured by reflection intensity

2.6 LOAM Matching

As the feature point selection in this work is built upon that introduced in LOAM, the relevant portions of LOAM are introduced in more detail here so that comparisons can be made later on.

LOAM is part of the ICP family of scan registration algorithms. It differs from ICP in that it uses a couple of assumptions about the scene in order to improve convergence, reduce incorrect correspondences, and to reduce the overall complexity of the problem. In

LOAM, the scene is assumed to have planar faces and geometric edges present. To find edges and planes, the approximate curvature of each point is calculated and points with low curvature are assumed to have been sampled from a planar face, and points with high curvature are assumed to be sampled from a geometric edge. The curvature is calculated as a 1-d convolution along each ring; a ring being the set of points from a single laser-detector. The lidar that collected Figure 2.1 has 32 clearly-defined rings.

Once the sets of planar feature points and edge feature points have been extracted for the current scan, the algorithm proceeds to iterate over matching them to the previous sets of feature points and then optimize the trajectory to minimize the matching error as in ICP. However, the costs themselves are modified to reflect the assumptions about the scene.

Each current edge feature point is corresponded to the nearest two edge points in the previous feature set. The previous two points are used as an implicit definition of a line, and the cost is the distance from the current edge point to the line. The cost associated with an edge feature and its corresponding points

$$d_{Edge}(P, P_A, P_B) = P - P_A + \frac{(P - P_A) \circ (P_B - P_A)}{(P_B - P_A) \circ (P_B - P_A)}(P_B - P_A) \tag{2.35}$$

where P is the active feature point and P_A, P_B are the nearest edge points in the previous feature set. The dot product operator is $\circ$.

Each current planar point is correspondence to the nearest three planar points in the previous feature set. The previous three points are used as an implicit definition of a plane, and the cost is the distance from the current planar point to the plane. The cost associated with a planar feature and its corresponding points

$$d_{Plane}(P, P_A, P_B, P_C) = \frac{((P_B - P_A) \times (P_B - P_C)) \circ (P - P_B)}{\|(P_B - P_A) \times (P_B - P_C)\|} \tag{2.36}$$

where the active feature point is P and P_A, P_B, P_C are the nearest planar points in the previous feature set.

Although LOAM's performance on the Kitti dataset is currently state-of-the-art, there are a few ways it might be improved:

- Firstly, the implicit geometry is parameterized by a minimal set of measured points. This is susceptible to noise due to the few points used and it is only by combining a robust loss with a large number of correspondences that LOAM reduces the impact of noisy implicit geometry.

- To correct for motion distortion from ego-motion, LOAM uses a hierarchical approach: IMU measurements are used to correct for any accelerations over the scan, then the transform parameterization is linearly interpolated for each point in the scan. Therefore, if an IMU is not present, LOAM is unable to correct for ego-motion due to ego-acceleration during scans, and must rely on linear interpolation.
- Finally, the intensity channel of the lidar is not used, ignoring potentially useful appearance information.

Chapter 3

Methodology

3.1 Design

3.1.1 High-Level Methodology

The proposed method uses LOAM [40] features with the addition of appearance features and a different scene model within a sliding window framework. LOAM is selected as a starting point for feature extraction because it works well with sparse LIDAR scans, and it performs well in structured environments despite having low complexity.

The principles that this method is based on are:

- Use the intensity channel to extract additional appearance information. This is intended to capitalize on road markings that are missed by the purely geometric features of LOAM. The intensity channel is used by looking for points having high intensity gradient along each ring in a lidar scan. Those points are modeled as being sampled from a collection of lines in order to fit linear road markings such as lane markers and pedestrian crosswalks.
- Apply the simultaneous trajectory estimation and mapping framework [1] (STEAM) to account for motion distortion. The STEAM framework is built on a continuous-time trajectory parameterization as well as a general motion model. Using a continuous-time parameterization removes the need to pre-correct each scan for distortion due to ego-motion. The motion model is flexible enough to not require an IMU to correct for ego-acceleration as LOAM does.

- Solve as a sliding window filter instead of having a separate mapping and odometry thread. Sliding window approaches tend to be more accurate than single-state filter approaches [10] due to the sliding window filter being less susceptible to accumulated linearization error. While LOAM does use a separate high-fidelity mapping thread to match the current scan to a map of accumulated feature points, only the trajectory during the current scan is optimized. Using a sliding window of several scans including the current scan could improve accuracy.

- Weight each correspondence based on a sensor noise model. An ideal lidar uses ideal lasers having zero beam width. Such a laser only measures the distance to a single point in the scene. In reality, the beams emitted from a lidar have a non-zero width, and each beam diverges as it travels further from the sensor. This means that there will be multiple surfaces encountered by the laser, each contributing to the reflected signal. This creates some ambiguity as far as which part of the reflected signal corresponds to the laser beam centerline.

  The ambiguity created by the divergence of the laser is modeled by assuming additive Gaussian noise on the bearing, azimuth and range of each measurement. This model leads to the Euclidean position of points measured further from the sensor having higher uncertainty. This noise model is expected to improve solution quality because it can be used to weight errors more or less depending on each point distribution.

- Use explicit geometric landmark states instead of minimally-defined implicit landmarks. LOAM does not explicitly parameterize lines or planes. Instead, the geometry is implicitly defined by a minimal set of feature points. Being a minimal set, this is sensitive to error in any of the individual points that define the geometry. Explicit geometry states can be fit to an arbitrary number of points. This can be done robustly, for example with RANSAC or robust loss functions, to remove the effect of outliers. Explicit landmark states also offer the possibility to be fit to feature points across an arbitrary number of scans, aiding when the sparsity of a single scan is insufficient to constrain a landmark.

3.1.2 High-Level Implementation

Although a lidar samples continuously, the proposed method processes lidar data grouped into scans with each scan corresponding to one revolution of the sensor. This simplifies evaluation because most lidar datasets are provided in this format.

The first step is to extract feature points from each scan by assigning each point in the scan a score representing how much that point fits the criteria to be a feature. The score is calculated based on either the range or intensity of each point and its nearest neighbours. Correcting for motion distortion is not necessary to extract feature points because the relative motion distortion of nearby points is small. Each point's score can be evaluated in parallel, so this step is relatively inexpensive.

Once feature points for the current scan have been found, the algorithm enters a loop:

- Find correspondences for the existing scene model in the set of feature points. This is implemented using kd-trees for efficient searching, and the quality of candidate correspondences is evaluated to remove those having high measurement error.
- The set of uncorresponded feature points from the current and previous scans is used to extend the scene model by initializing new landmarks. The distribution and number of landmarks is controlled by binning them according to their location in the scene and enforcing a limit for each bin.
- The correspondences between the feature points and scene model are converted into error residuals for the optimizer.
- Error residuals are created for the motion model.
- Nonlinear optimization is used to minimize feature residuals and motion residuals by adjusting the landmark and trajectory states.

The loop terminates when the optimized state changes less than a threshold or a maximum number of iterations is reached.

Algorithm 1 summarizes the overall processing steps for a single lidar scan. Advancing the sliding window happens outside of this function and is described later.

Algorithm 1 Odometry

```
1: procedure ProcessScan(scan, state, landmarks)
2:     featurePoints ← extractFeaturePoints(scan)
3:     while notconverged do                    ▷ Based on norm of state updates
4:         correspondences ← correspondLandmarks(landmarks, featurePoints)
5:         landmarks ← landmarks + newLandmarks(featurePoints, correspondences)
6:         featureResiduals ← createFeatureResiduals(correspondences, featurePoints, landmarks)
7:         motionResiduals ← createMotionResiduals()
8:         state ← optimizeState(featureResiduals, motionResiduals)
```

3.1.3 Scene Model

The scene is explicitly modeled as a collection of geometric edges, planes, and intensity edges on planes. For each point in a scan, a curvature score is calculated and used to select points that could be either a geometric edge or plane. The geometric score is the same as in LOAM, except that it is not normalized by range. The scene is assumed to be static.

Intensity feature points are constrained to have low geometric curvature and low local curvature variance before being selected as features. This reduces features selected from noisy parts of the scene, particularly vegetation.

Both scores are computed as a 1-dimensional convolution for each laser scan line. This is because the sensor has finer azimuthal resolution compared to elevation. Features are spread across angular bins to distribute them throughout the scan and the density of feature points is limited to avoid oversampling one region in the scene.

Feature Point Definition

The feature points are extracted from each laser line individually, where a laser line consists of the points measured by a single laser-detector pair on the lidar. The input scan is defined as $\mathbb{M} = \{M_1, ..., M_N\}$, $M_i = \{m_1, ..., m_n\}$, $m_i = \{p_i, l_i\}$, $p_i \in \mathbb{R}^3$, $l_i \in \mathbb{R}$ where N is the number of laser lines, m_i is a measurement returned by the lidar, p_i is a point measured by the lidar, and l_i is the intensity measurement of that point.

The scoring functions used to select feature points take either the range or intensity signals of each ring in the scan. The range signal is defined as $\mathbb{D} = \{D_1, ..., D_N\}$, $D_i = \{r_1, ..., r_n\}$, $r_j = \sqrt{\|p_j\|^2}, \forall r_j \in D_i, \forall D_i \in \mathbb{D}$ and the intensity signal is $\mathbb{B} = \{B_1, ..., B_N\}, B_i = \{b_1, ..., b_n\}$ where b_j is the intensity measurement for an individual point. Histogram equalization is used to pre-process raw intensity readings in order to increase contrast. It is possible to adjust lidar intensity based on the range of the point, the angle of incidence, and an intrinsic intensity calibration for each laser, but this was not found to be necessary because lane markings are mildly retro-reflective and generate relatively reliable high intensity measurements compared to other surfaces. Furthermore, as the scores are calculated on each ring individually, any intensity offsets between any pairs of rings will not affect the scores.

The curvature score is computed with 1-dimensional discrete correlations between each element in $\mathbb{D}$ and the kernel plotted in Figure 3.2. This is similar to the scoring function in [40], with the difference being that the score here is not normalized by range.

The intensity score is similarly computed with correlation between each element in $\mathbb{B}$ and the kernel plotted in Figure 3.3. This kernel was designed to pick up the high intensity gradients present between lane markings and asphalt. It is similar to the Sobel operator except that it is wider to provide some resilience to noise through an averaging effect. The drop-off in the magnitude of the weights at the edges of the kernel is done to better handle short intensity features spanning less than half the kernel. If the magnitude of the weights were the same, then the kernel would produce the same score for multiple points, rather than assigning the highest score to the point closest to the intensity gradient. This condition is encountered in autonomous driving when thin lane markings are scanned.

The filtering steps described in [40] are applied here to remove points with invalid scores due to occlusions, missing data, or degenerate scene geometry. In addition, points having high range variance or high curvature are invalidated as potential intensity edges.

The sign of the scores is used to aid correspondence between feature points. For intensity gradients, the score will change sign depending on if the gradient is from high to low or low to high. A similar sign change occurs between geometric edges depending on if they have positive or negative curvature. Therefore, distinguishing feature points by sign reduces the risk of making incorrect correspondences.

There are 5 different feature point definitions used in this work:

- $\mathbb{S}_{E+}$ Geometric edges, selected for high curvature score
- $\mathbb{S}_{E-}$ Geometric edges, selected for low curvature score
- $\mathbb{S}_{F}$ Geometric planes, selected for near zero curvature score
- $\mathbb{S}_{B+}$ Intensity edges, selected for high intensity gradient
- $\mathbb{S}_{B-}$ Intensity edges, selected for low intensity gradient

Once all the scores are calculated, the points are sorted by each of the criteria, selecting the best scoring points for each type of feature definitions. To spread the points throughout the scene, the scan is broken into radial-azimuthal bins, and the highest scoring points from each bin are selected. The distance between feature points must also be a greater than a threshold to avoid picking too many feature points from high-density regions of the scan.

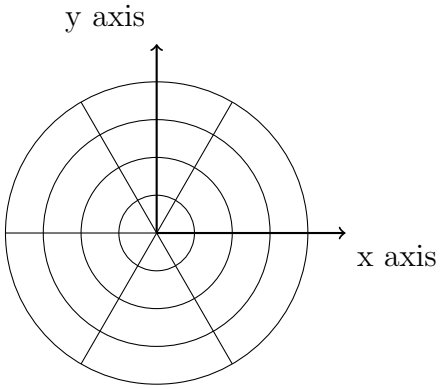

Figure 3.1: Example sectors for distributed feature point sampling

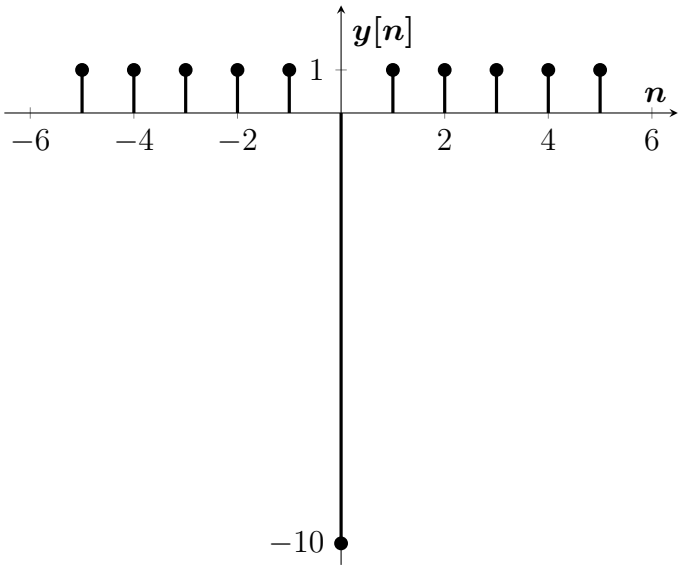

Figure 3.2: Plot of curvature scoring kernel

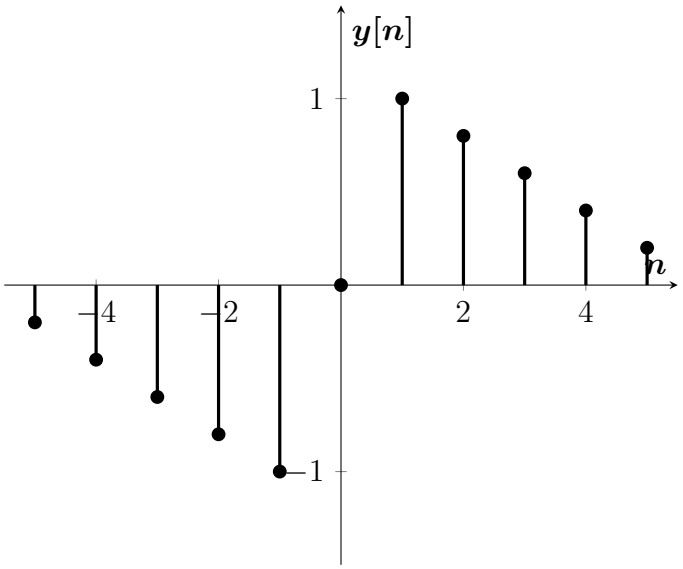

Figure 3.3: Plot of intensity scoring kernel

A collection of geometric feature points extracted from a typical scan is shown in Figure 3.4. This figure demonstrates the edge features are found on poles and the planar features found on the ground, as well as how planar features are distributed evenly throughout the scan, rather than concentrated in the flattest part of the ring.

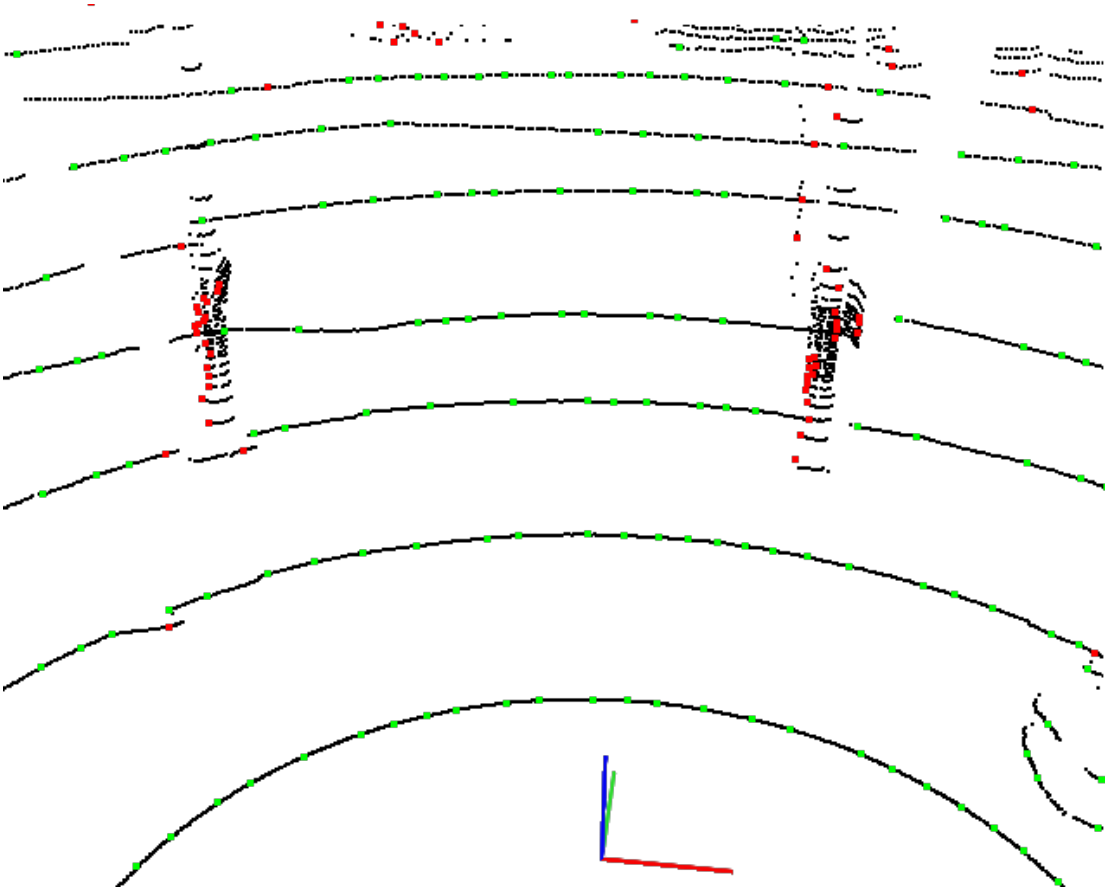

Figure 3.4: Visualization of feature extraction on a sample scan. The sensor is located at the origin. The black points are the original scan, the red points are edge points, and the green points are planar points.

Landmark Definition

Landmarks are defined as being either edges or planes. An edge landmark is parameterized by a 3-D point on the edge and a direction. Edges are defined by

$$\text{Edge } (P_0 \in \mathbb{R}^3, \hat{n} \in \mathbb{S}^2) := \mathbb{L}^3 = \{p \in \mathbb{R}^3 : p = P_0 + \hat{n}x, x \in \mathbb{R}\} \tag{3.1}$$

A plane landmark is defined as

$$\text{Plane } (P_0 \in \mathbb{R}^3, \hat{n} \in \mathbb{S}^2) := \mathbb{P}^3 = \{p \in \mathbb{R}^3 : (p - P_0) \circ \hat{n} = 0\} \tag{3.2}$$

Planes are parameterized by a 3-D point and a plane normal. Note that a line has 4 degrees of freedom and a plane has only 3 degrees of freedom due to the constraints.

For the plane normal and the edge direction, the magnitude of the vector does not change the landmark. To remove this extraneous degree of freedom the direction is constrained to be on $\mathbb{S}^2$ or the unit sphere.

Each feature point is treated as a measurement of a corresponding landmark. A typical collection of landmarks is visible in Figure 3.5 against a scan being matched to those landmarks. Two dimensional cyan boxes are drawn centered on the point partially parameterizing the plane, while red lines are drawn centered on the point partially parameterizing the edge. Each line is shown with a uniform length for visibility, but the spread of the feature points corresponding to each edge may not extend all the way along the drawn line. Edge landmarks tend to be placed on building corners and poles, while planes are fit to walls and the ground. Not all feature points have a corresponding landmark depending on how many landmarks are used to model the scene. Some surfaces in the scene are relatively complex, which can result in the algorithm attempting to initialize multiple landmarks in close proximity to one another. Due to each landmark being free to change during optimization as well as the optimization making use of a robust cost function, these clustered landmarks do not cause the algorithm to fail.

Landmark State Optimization

While n-dimensional unit spheres in general are not Lie Groups, there exists an exponential map between an element $p \in \mathbb{S}^n$ and an element of the tangent hyperplane $v \in T_p\mathbb{S}^n$ to another element in $\mathbb{S}^n$ [16]. The tangent plane of p is

$$T_p\mathbb{S}^n := \left\{ v \in \mathbb{R}^{n+1} : \quad p \circ v = 0 \right\} \tag{3.3}$$

where $\circ$ is the dot product operator. The exponential map for element p with a tangent quantity v is

$$\exp_p v := \begin{Bmatrix} \cos(\|v\|)p + \sin(\|v\|)\dfrac{v}{\|v\|}, & v \in T_p\mathbb{S}^n \setminus \{0\}, \\ p, & v = 0 \end{Bmatrix} \tag{3.4}$$

The Jacobian of the exponential map approaches identity as the tangent quantity approaches zero. This is a reasonable approximation for small tangent quantities assuming that the step size of the optimization approaches zero as the problem converges.

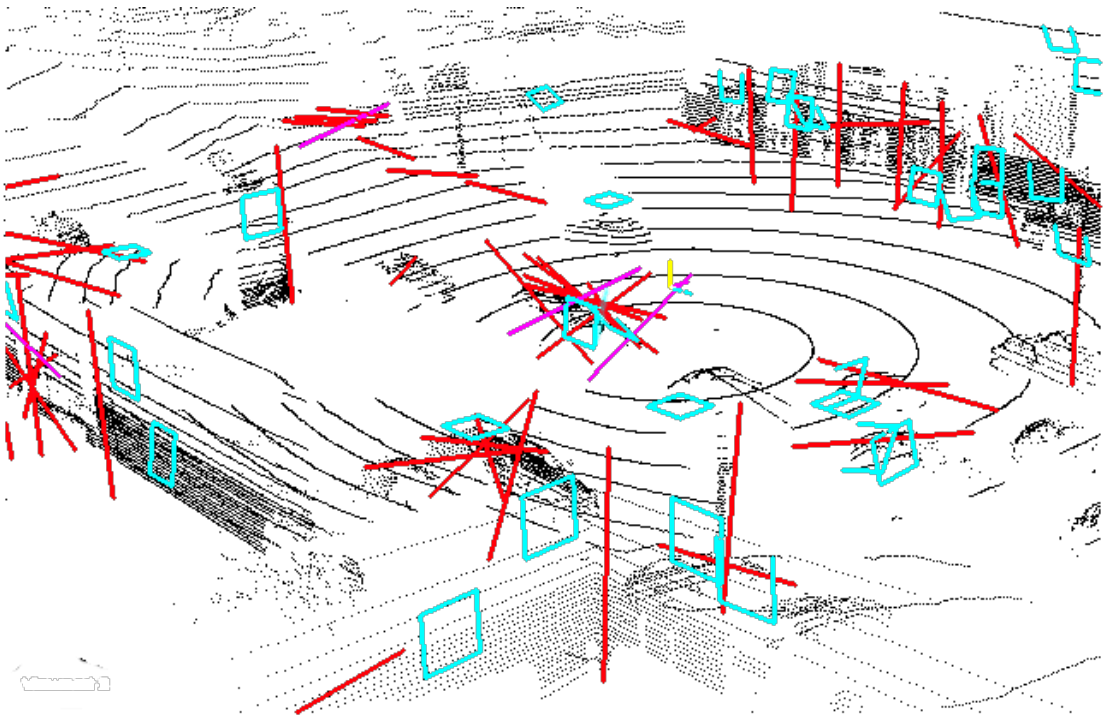

Figure 3.5: Visualization of landmarks against a lidar scan. Geometric edges are red, geometric planes are cyan.

The tangent hyperplane at any point in $\mathbb{S}^n$ has only n degrees of freedom due to the constraint in Eq. 3.3. Constrained optimization should be avoided where possible because it requires extra care that the constraints are met, and because the Jacobian with respect to an over-parameterized representation may cause a solver to attempt to move in a direction that violates the constraints, slowing or even stalling convergence.

Optimizing spherical elements is done in similarly to optimizing elements of SE(3): a minimally parameterized perturbation is optimized instead of the element itself. For SE(3), the perturbation is defined in the tangent space of the identity element. In order to use this same idea for spheres, a specific tangent space $T_P\mathbb{S}^n$ corresponding to element

$$P \in \mathbb{S}^n : P_i := \begin{Bmatrix} 0, & i \neq n+1 \\ 1, & \text{otherwise} \end{Bmatrix} \tag{3.5}$$

is chosen to hold the perturbation.

Given the definition of $T_P\mathbb{S}^n$, it is clear that a trivial linear map from $\mathbb{R}^n \to T_P\mathbb{S}^n$ exists. The map

$$\begin{aligned} f(x) &: \mathbb{R}^n \to T_P\mathbb{S}^n \\ f(x) &:= \begin{bmatrix} I_n \\ 0 \end{bmatrix} x \end{aligned} \tag{3.6}$$

where I_n is the n-dimensional identity matrix will be referred to as the expansion map.

A rotation map between $T_P\mathbb{S}^2 \to T_p\mathbb{S}^2$ is required to apply the optimized perturbation to any $p \in \mathbb{S}^2$ using the exponential map. The rotation matrix is that which rotates P onto p. The axis of rotation

$$\vec{w} = P \times p \tag{3.7}$$

and the magnitude of rotation

$$\cos\theta = P \circ p \tag{3.8}$$

are converted to a rotation matrix using the Rodrigues' rotation formula,

$$R = I + \vec{w}^{\wedge} + \vec{w}^{\wedge}\vec{w}^{\wedge}\frac{1}{1+\cos\theta} \tag{3.9}$$

and is valid for $p \in \mathbb{S}^2 \setminus \{-P\}$.

With the rotation matrix defined, the rotation mapping

$$\begin{aligned} g(x) &: T_P\mathbb{S}^2 \to T_p\mathbb{S}^2 \\ g(x) &= Rx \end{aligned} \tag{3.10}$$

is straightforward.

The rotation map is not defined when the rotation between p and P is exactly 180 degrees. In this application, the magnitude of the feature errors (Eqs 3.15 & 3.16) does not change if $\hat{n}$ reverses. This is exploited to keep $\hat{n}$ in the upper hemisphere of $\mathbb{S}^n$ by multiplying any $\hat{n}$ having a negative z component by -1 after the update step in the optimizer. Restricting the domain to the upper hemisphere sidesteps the singularity because any $\hat{n}$ will be at most 90 degrees away from P.

The $\boxplus$ operator was introduced to denote the operation composing an element of SE(3) with an element of se(3) in Equation 2.6. The operator will be overloaded here as needed to denote any mapping $E \times T \to E$ where E is some over-parameterized set and T is a set containing minimally-parameterized perturbations of elements in E.

An update scheme for a spherical element $p \in \mathbb{S}^2$ given a perturbation $\delta p \in \mathbb{R}^2$ is created by using $f(x)$ for $n = 2$, the rotation map $g(x)$, and the exponential map

$$\begin{aligned} p' &= p \boxplus \delta p \\ p' &= exp_p(g(f(\delta p))) \end{aligned} \tag{3.11}$$

where $p' \in \mathbb{S}^2$ is the updated spherical element.

The Jacobian of the update scheme

$$\frac{\partial(p')}{\partial(\delta p)}\Big|_{\delta p=0} = R \begin{bmatrix} I_2 \\ 0 \end{bmatrix} \tag{3.12}$$

is taken for $\delta p = 0$ because the perturbations will approach zero as the optimization converges, so it is a reasonable simplification for implementation.

Both the line and plane parameterizations contain a spherical quantity $\hat{n}$ that is optimized using the scheme just introduced. However, P_0 as used in the landmark parameterizations is also over-parameterized and requires an update scheme to use unconstrained optimization.

For line landmarks, P_0 as defined in Eq. 3.1 has 1 extra degree of freedom: moving P_0 along $\hat{n}$ does not change the line. Given an update $\delta P_0 \in \mathbb{R}^2$ the update scheme

$$P_0' = P_0 + R \begin{bmatrix} I_2 \\ 0 \end{bmatrix} \delta P_0 \tag{3.13}$$

where R is the rotation matrix used in the spherical update scheme removes the extra degree of freedom by constraining P_0 to only move orthogonally to direction of the line. Together, this update scheme and the spherical update scheme form the line update scheme.

For planes, P_0 has 2 extra degrees of freedom arise because P_0 may be moved anywhere in the plane without changing the plane itself. Given an update $\delta P_0 \in \mathbb{R}$ the update scheme

$$P_0' = P_0 + \delta P_0 \hat{n} \tag{3.14}$$

where $\hat{n} \in \mathbb{S}^2$ is the plane's normal vector removes the extra degrees of freedom by constraining P_0 to only move parallel to the plane normal. Together, this update scheme and the spherical update scheme form the plane update scheme.

Using the line and plane update schemes introduced allows the landmarks to be optimized using regular unconstrained optimization.

3.1.4 Feature Cost

Errors are formed between each landmark state and any corresponded feature points. The error is the Euclidean distance between the feature point and the closest point on the geometric landmark. The error for an edge feature point is

$$\vec{e}_{i,E} = (P_i - P_0) - ((P_i - P_0) \circ \hat{n})\hat{n} \tag{3.15}$$

where P_i is the feature point is the distance from the point to the line. This is calculated as the $\mathbb{R}^3$ translation between the feature point and the line in the landmark frame.

The error for a planar feature point is

$$e_{i,P} = (P_i - P_0) \circ \hat{n} \tag{3.16}$$

where P_i is again the feature point is the distance from the point to the plane.

The optimization is formulated as a least-squares problem in order to take advantage of the mature nonlinear least-squares solvers available. In order to use these solvers, the Jacobian of the errors is required and must be continuous. In this case, the cost functions are simple enough that the Jacobians are found by inspection. The Jacobians for the line

error are with respect to P_i, P_0 and to $\hat{n}$.

$$\begin{aligned}
\frac{\partial \vec{e}_{i,E}}{\partial P_i} &= I - \hat{n}\hat{n}^T \\
\frac{\partial \vec{e}_{i,E}}{\partial P_0} &= -I + \hat{n}\hat{n}^T \\
\frac{\partial \vec{e}_{i,E}}{\partial \hat{n}} &= -\hat{n}(P_i - P_0)^T - ((P_i - P_0) \circ \hat{n})I
\end{aligned} \tag{3.17}$$

The Jacobians of the plane error are also with respect to P_i, P_0 and to $\hat{n}$.

$$\begin{aligned}
\frac{\partial e_{i,P}}{\partial P_i} &= \hat{n}^T \\
\frac{\partial e_{i,P}}{\partial P_0} &= -\hat{n}^T \\
\frac{\partial e_{i,P}}{\partial \hat{n}} &= (P_i - P_0)^T
\end{aligned} \tag{3.18}$$

The edge error being a vector is suitable for the least-squares solver used. However, the error is over-parameterized because the point-to-line distance only has one degree of freedom. It is possible to reduce the dimensionality by rotating the error into a new basis with one of the basis vectors parallel to the original error vector. This was experimented with to determine if reducing dimensionality could speed up the solver, but it was found to add a significant number of iterations, likely because as the error approaches zero, its direction becomes undefined.

Error Whitening & Robust Loss

The points returned by the lidar used in this work are returned in spherical coordinates: range, azimuth, and elevation. The points are modeled as samples of random variables with independent Gaussian distributions, with known covariance Σ_S. The data from the sensor is transformed from spherical to Euclidean coordinates using

$$\begin{aligned}
x &= r\cos\theta\cos\phi \\
y &= r\cos\theta\sin\phi \\
z &= r\sin\theta
\end{aligned} \tag{3.19}$$

where the variables are defined in Figure 3.6.

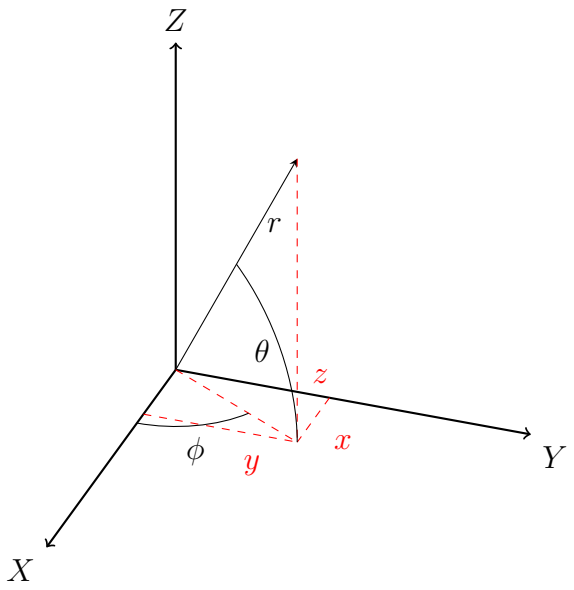

Figure 3.6: Spherical to Euclidean Coordinate Transform

The spherical covariance is transformed to a Euclidean covariance

$$\Sigma_E \approx J^T \Sigma_S J \tag{3.20}$$

where J is the Jacobian of Equation 3.19 and Σ_E is the Euclidean covariance. This is a first order approximation.

Starting from the Euclidean covariance, a covariance estimate for each error can be calculated in a similar manner as shown in Equation 3.21. The estimate is used to whiten the error: $e_w = \Sigma_e^{-\frac{1}{2}} e$. The weight takes into account the certainty of far points compared to near points, as well as the orientation of Σ_E compared to the landmark. It is assumed that the orientation of the landmark and location of the measured point will not change significantly during optimization, which is reasonable provided the optimization has good initialization. Under this assumption the covariance estimate will also not change significantly. This is taken advantage of by not updating the covariance estimate between optimization iterations, in order to avoid adding complexity to the Jacobian of the error.

$$\Sigma_e \approx J_P^T \Sigma_E J_P \tag{3.21}$$

The total feature error E_f formulated as the sum of the individual squared, whitened feature errors.

$$E_f(E_E, E_P) = \sum_{i=0}^{N} \vec{e}_{w,i,E}^{T} \vec{e}_{w,i,E} + \sum_{i=0}^{M} e_{w,i,P}^{2} \tag{3.22}$$

where E_E is the set of N edge feature points with landmark correspondences, E_P is the set of M plane feature points with landmark correspondences, and all errors have been whitened. Minimizing this cost is a standard least-squares problem, but a least-squared formulation is not suitable for this applications due to the chance of incorrect correspondences.

Outliers caused by incorrect correspondences have a large impact on the overall solution when using a least-squares formulation. To avoid this problem, rather than using a least-squares cost, a Cauchy loss is applied. The Cauchy loss function completely removes the impact of large errors, compared to the commonly-used Huber loss which merely reduces the impact of large errors. This has increased robustness to outliers at the cost of poor convergence if the problem is initialized far from the minimum.

The Cauchy loss function

$$\mathcal{C}(e, k) = \begin{cases} \frac{k^2}{6}(1 - (1 - \frac{e^T e}{k^2})^3) & e^T e \leq k^2 \\ \frac{k^2}{6} & \text{otherwise} \end{cases} \tag{3.23}$$

where k is a parameter used to adjust the cutoff point beyond which the error does not affect the optimization, and e is the error vector. Although it is a piecewise function, it is of class C^2. The landmarks are robustly fit, courtesy of the Cauchy loss, to any corresponded feature points.

The modified total feature cost $\mathcal{E}_f$

$$\mathcal{E}_f(E_E, E_P, k) = \sum_{i=0}^{N} \mathcal{C}(\vec{e}_{w,i,E}, k) + \sum_{i=0}^{M} \mathcal{C}(e_{w,i,P}, k) \tag{3.24}$$

3.1.5 State Formulation

The STEAM framework is used because a rotating LIDAR continuously samples its environment. A traditional discrete-state SLAM formulation has a different state for each measurement time. In this application, this could amount to thousands of largely redundant states because each point is measured at a different time. Instead, this framework

provides a continuous representation of a trajectory with relatively few parameters: the state consists of pose and body-centered velocity at times spaced evenly across the optimization window, and the state at any time within the window is a function of those discrete states.

It is possible to use a continuous formulation built upon temporal parametric basis functions, but the choice of basis function is not coupled to the true motion model. In contrast, formulating a motion model as a Gaussian process as in [4] results in an interpolation scheme that is consistent with the physical motion.

The set of trajectory states $\mathcal{T}$ is defined in Equation 3.25, where N is the number of states, ϖ is the body centered velocity expressed as the concatenation of angular velocity $w \in \mathbb{R}^3$ and linear velocity $v \in \mathbb{R}^3$, and subscripts follow the convention introduced in section 2.1. The magnitude of w is the rate of rotation about the unit vector coincident with w. Frame O is an inertial frame that provides a reference for all the states in the window. The lidar sensor frames are labeled by their position within the window, the first sensor frame is frame 1, and the last is frame n.

$$\mathcal{T} = \left\{ \{T_{O1}, {}_1\varpi_{O1}\}, \quad ..., \quad \{T_{On}, {}_n\varpi_{On}\} \right\} : T_{Oi} \in \mathrm{SE}(3), {}_O\varpi_{1O} \in \mathbb{R}^6 \tag{3.25}$$

The poses within an optimization window of size 4 are visualized in Figure 3.7. The frames are labeled in the top portion of the Figure, while the transforms are shown as directed arrows that indicate which frames each transform moves between. The common portion of the transform subscripts is omitted to save space. The position of frame O with respect to the sensor frames is a gauge freedom characteristic of SLAM: the position of O can be set arbitrarily without changing the total error in the system. To remove the gauge freedom, the first transform in the window, T_1, is defined as identity and not optimized.

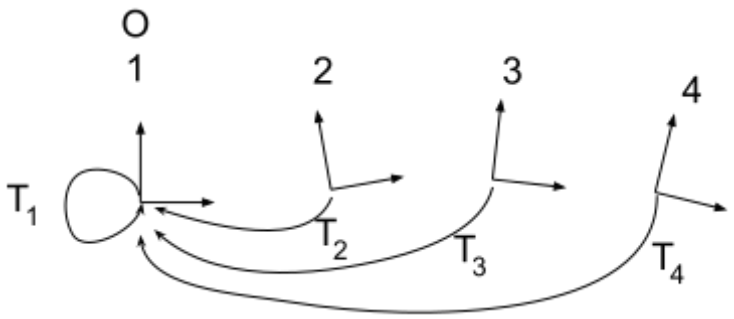

Figure 3.7: Visualization of poses within the optimization window

In this odometry formulation $\mathcal{T}$ is only part of the global set of states. There are also states parameterizing the landmarks. Let G be the number of edge landmarks with corresponding edge feature points, and H be the number of plane landmarks with corresponding plane feature points. Then $\mathcal{L} = \{l_1, ..., l_G\} : l_i \in \mathbb{L}^3$ is the set of all edge landmarks and $\mathcal{R} = \{r_1, ..., r_H\} : r_i \in \mathbb{P}^3$ is the set of all plane landmarks. All landmarks are defined in frame O. The complete set of states is $\mathcal{S} = \{\mathcal{T}, \mathcal{L}, \mathcal{R}\}$.

Jacobian of Interpolated Transform

Although [4] provides the framework for interpolating transforms using Gaussian processes, the Jacobian of the interpolated transform with respect to perturbations on the state variables is not explicitly stated, and it is required in order to integrate the approach into the solver library selected.

The derivation result can be summarized by stating that the perturbation on any interpolated transform at time t_i can be approximated as a linear combination of the perturbations on the state variables at times t_k, t_{k+1} where $t_k \leq t_i \leq t_{k+1}$. The approximation is valid for small perturbations of the state variables, and is reasonable given that the perturbations will approach zero as the optimization converges. Appendix A provides the derivation for

$$\epsilon_i \approx A_1(T_k, \varpi_k, T_{k+1}, \varpi_{k+1}, t_i) \begin{bmatrix} \epsilon_k \\ \psi_k \end{bmatrix} + A_2(T_k, \varpi_k, T_{k+1}, \varpi_{k+1}, t_i) \begin{bmatrix} \epsilon_{k+1} \\ \psi_{k+1} \end{bmatrix} \tag{3.26}$$

where ϵ are perturbations on state transform variables, ψ are perturbations on state twist variables, and A_1, A_2 are interpolation factors that are functions of the states at times t_k, t_{k+1} and t_i.

Jacobian of Point Transformation

As all the feature points must be transformed into the frame of the landmarks, the Jacobian for each transformed point with respect to the transform is required.

Points are transformed from one frame to another using

$${}_ip_1 = (T_{ij} \boxplus \epsilon)_j p_1 \tag{3.27}$$

where ϵ is a twist perturbation on the transform T_{ij}. This function has the Jacobian

$$\frac{\partial_i p_1}{\partial \epsilon} = [-{}_ip_1^\wedge : I] \tag{3.28}$$

3.1.6 Data Association

Data association is required when corresponding feature points to existing landmarks. A feature point is considered to be corresponding to a specific landmark when the feature error given the point and the landmark is the lowest among all possible combinations of landmark and feature point, and the error is below a threshold.

Let p be a feature point associated with landmark set Q. The corresponding landmark is $q_k \in Q$ and is subject to

$$f(p, q_k, \mathcal{T}) \leq f(p, q_i, \mathcal{T}) \, \forall q_i \in Q, f(p, q_k, \mathcal{T}) \leq \lambda \tag{3.29}$$

where f is the error function given the correspondence, $\mathcal{T}$ is the trajectory state, and λ is the error threshold. If a correspondence that has an error below the threshold does not exist, that feature point is considered as not having a correspondence and is not used. The error function is either point-to-line or point-to-plane distance depending on the feature point type.

The optimal state $\mathcal{S}^*$ given a set of correspondences C is found by solving

$$\mathcal{S}^* = \underset{\mathcal{S}}{\operatorname{argmin}}(\mathcal{E}_f + \mathcal{E}_m) \tag{3.30}$$

where $\mathcal{E}_f$ and $\mathcal{E}_m$ are the feature error and motion model error previously introduced.

Finding correspondences and solving Eq. 3.30 is iterated until the correspondence set does not change or until the number of iterations reaches a limit. The limit is used to bound the total number of iterations so as to have predictable runtime.

This data association technique is quite similar to that of ICP, the only difference is that the error is formulated as point to landmark distance rather than point to point distance. The impact of incorrect correspondence is reduced through the use of the robust loss function applied in the feature error.

3.1.7 Sliding Window Update

In a sliding window filter, the state at the start of the window is removed whenever a new state is added to the end of the window in order to maintain a constant window size. Traditionally, states that are removed from a sliding window filter are marginalized out. Marginalization converts the removed states into linear costs on states still in the window.

However, marginalization changes the structure of the problem such that it can no longer be solved as efficiently. Therefore, for a given computational budget, there is a choice between using marginalization or not using marginalization but enjoying a significantly larger window size. The proposed algorithm opts for the latter choice because other works have found that using a larger window size without marginalization can be more accurate and because not using marginalization simplifies the implementation [36].

The trajectory states in the window are defined with respect to frame O, which is coincident with frame 1. Therefore, when states are removed from the window, the remaining states must be redefined to be with respect to the new start of the window.

Let frame k be the new start of the window. Frame O', the new inertial reference frame is coincident with frame k. Transform T_{k1} is equivalent to transform $T_{O'O}$, and is readily available in the state window. All states that will remain in the window $\mathcal{T}^* = \{\{T_{T_{Ok}}, {}_O\varpi_{Ok}\}, ..., \{T_{T_{On}}, {}_O\varpi_{On}\}\}$ converted to be with respect to frame k is the set $\mathcal{T}' = \{\{T_{T_{O'k}}, {}'_O\varpi_{O'k}\}, ..., \{T_{T_{O'n}}, {}'_O\varpi_{O'n}\}\}$. The transforms are converted using transform concatenation defined in Eq. 2.2 and the body-centered velocities are transformed using

$$ {}_{O'}\varpi_{O'k} = \begin{bmatrix} R_{O'O} & 0 \\ 0 & R_{O'O} \end{bmatrix} {}_O\varpi_{Ok} \tag{3.31} $$

where $R_{O'O}$ is the rotational component of $T_{O'O}$.

The landmarks are also defined in frame O. Consequently when a state is removed from the front of the window, the landmarks must be transformed into frame k. All landmarks are parameterized by a point and a vector, and so are transformed the same way regardless of landmark type. That transform is

$$ \begin{bmatrix} \hat{n}' \\ P'_0 \end{bmatrix} = \begin{bmatrix} R_{O'O} & 0 \\ 0 & R_{O'O} \end{bmatrix} \begin{bmatrix} \hat{n} \\ P_0 \end{bmatrix} + \begin{bmatrix} 0 \\ {}_{O'}t_{O'O} \end{bmatrix} \tag{3.32} $$

where $(\hat{n}, P_0)$ is the landmark parameterization with respect to frame O, $(\hat{n}', P'_0)$ is the same landmark with respect to frame O', and $(R_{O'O}, {}_{O'}t_{O'O})$ is $T_{O'O}$.

3.2 Implementation Details

This section documents the solutions used for some hitherto unmentioned pieces of the algorithm. These solutions are independent from the main algorithm in that they could be replaced by alternatives without affecting the design in Section 3.1.

The entire algorithm is implemented in `C++`, and source code is available here as the wave_odometry module.

3.2.1 Landmark Merging

If the scene contains large planar areas or long edges, it is possible that multiple landmarks may be redundantly modeling the same features in the environment. For example, straight curbs and planar patches on the ground tend to have multiple landmarks created when the model is fit to the scene. A scheme to merge redundant landmarks is therefore deployed to remove unneeded complexity. In addition redundant landmarks can be problematic during data association, because multiple similar landmarks can slow convergence as data associations are more likely to change between iterations.

In order to decide whether or not a given pair of landmarks should be merged, a distance is calculated between all nearby landmarks. Nearby landmarks are found only considering the point portion of the landmark parameterization and using a kd-tree seach.

A distance is calculated between each landmark and the set of landmarks that are nearby the first landmark. The distances are sorted, then if the lowest distance is below a threshold, the landmarks are merged by averaging the landmark parameterizations.

For line landmarks, the distance score is a weighted combination of the line-to-line distance and angular difference between the line directions. For planes, it is a weighted combination of the distance between P_0 as defined in Equation 3.2 and the angular difference between the normals.

3.2.2 Landmark Initialization

Landmarks must be created when starting or when the robot has moved far enough in the scene that the existing set of landmarks does not have sufficient correspondence with the current set of feature points to constrain the most recent state. The problem of landmark initialization is to create landmarks with a good distribution throughout the scene and to initialize the parameters of those landmarks close to the optimal values so that when optimizing, the parameters will converge to their optimal values.

Plane Landmark Initialization

Let $\mathcal{S}_{F0}$ be the set of geometric plane feature points extracted from the current scan, and $\mathcal{S}_{F1}$ be the set of geometric plane feature points extracted from the previous scan. Planes

are initializing by finding the 3 nearest neighbours in $\mathcal{S}_{F1}$ for each element in $\mathcal{S}_{F0}$ after both sets of points have been transformed to frame O using the current trajectory estimate. The nearest neighbour set is $K = \{\{p_1, p_2, p_3\}_1, ..., \{p_1, p_2, p_3\}_n\}$ where each element of K corresponds to a feature point in $\mathcal{S}_{F0}$.

A plane parameterization is created from each element of K by converting the 3-point parameterization to the point and normal parameterization already introduced using

$$\begin{aligned} &\vec{v}_1 = p_2 - p_1, \quad \vec{v}_2 = p_2 - p_3 \\ &\hat{n} = \frac{\vec{v}_1 \times \vec{v}_2}{\|\vec{v}_1 \times \vec{v}_2\|}, \quad P_0 = \frac{\sum_{i=1}^{3} p_i}{3}, \quad \{P_0, \hat{n}\} \in \mathbb{L}^3 \end{aligned} \tag{3.33}$$

The error from each initialized plane to the corresponding element in $\mathcal{S}_{F0}$ is calculated and if less than a threshold, is added to the set of all initialized planes $\mathcal{R}^*$.

It is preferred to create landmarks for the largest planar surfaces in the scene because they have a greater chance of being sampled in subsequent scans. Landmark merging is done on $\mathcal{R}^*$ as described in Section 3.2.1 to produce the merged set $\bar{\mathcal{R}}$. The set $\bar{\mathcal{R}}$ is sorted in order of corresponding feature points because the elements of $\bar{\mathcal{R}}$ having many corresponding feature points are elements that have been produced by merging multiple elements of $\mathcal{R}^*$, and have likely been sampled from larger planes in the scene.

A quota for new landmarks is used in order to limit the number of redundant landmarks. The scene is divided into range-azimuth bins as with feature point extraction, and each range-azimuth bin is further divided into plane-normal bins. There is a limit imposed on each combination of range-azimuth-normal.

Before new planes are created, correspondences in the existing set of plane landmarks $\mathcal{R}$ are found in $\mathcal{S}_{F0}$. Each landmark with a corresponding point in $\mathcal{S}_{F0}$ increases the count in the appropriate range-azimuth-normal bin. This prevents new planes being created over top of existing planes.

The last step is to iterate over $\bar{\mathcal{R}}$ moving each element into $\mathcal{R}$ and increasing the count in the corresponding range-azimuth-normal bin, unless that bin has already reached its limit. This continues until $\bar{\mathcal{R}}$ is exhausted or all the bins are full.

Line Landmark Initialization

Initialization of line landmarks is done differently than plane landmarks, but it follows the same principle of creating a set of lines from minimal sets of feature points, then merging them to find lines defined by many feature points.

Starting with a set of edge points $E = \{e_1, ..., e_n\} : e_i \in \mathbb{R}^3$, the first step is to find the k nearest neighbours $C_i = \{c_1, ..., c_k\} : c_j \in \mathbb{N}$ for each e_i meeting the criteria

$$C_i = \{c_1, ..., c_k \; | \|e_i - e_{c_j}\| \leq \|e_i - e_l\|, \forall c_j \in C_i, l \notin C_i\} \tag{3.34}$$

For each element $e_i \in E$, each permutation of two elements from C_i is used to define a line, and the distance from that line to e_i is calculated. If the distance is small, then that line is placed into a set L_i. Once all possible combinations have been explored, similar lines within L_i are merged. The set is then searched for the line having the most number of edge points and moved into the set of initialized lines $\mathcal{L}^*$ if the number of edge points is greater than a threshold. Well-defined edges have enough sample points that multiple line permutations from C_i result in virtually the same line, so they will get merged. This algorithm rejects noisy edge points because it is unlikely that noisy edge points will have multiple permutations within C_i that produce the same line.

Once E is exhausted, the merging procedure is run on $\mathcal{L}^*$ to produce set $\bar{\mathcal{L}}$. The same binning procedure as that used when creating new plane landmarks is used to move lines from $\bar{\mathcal{L}}$ to $\mathcal{L}$.

3.2.3 Solving

Equation 3.30 is solved using the Levenberg-Marquardt algorithm as implemented in the Ceres solver library [3]. Ceres is used because it contains a Schur's complement based factorization that is particularly suited to bundle adjustment, and the odometry formulation presented in this work has a similar structure to bundle adjustment. Ceres also contains QR decomposition and Cholesky based factorization, but these were found to be slower in this application.

Trajectory State Initialization

As solving Equation 3.30 is the dominant computational cost, it is vital that the states are initialized as close as practical to the optimal point in order to limit the number of iterations. Good initialization is also important because the proposed method is not designed with a large convergence basin as a goal.

When operating in steady-state, the motion model can be used with existing trajectory states to generate reasonable initialization for new trajectory states. However, when first starting, existing trajectory states do not exist, making initialization more challenging.

As this work was developed with automotive applications in mind, initialization at startup is considered outside the scope of the problem. For automotive applications, simply using the wheel speed sensors to provide an initial velocity estimate would suffice to initialize the state within the convergence basin. Therefore, when evaluating on datasets, the velocity at startup is provided by ground truth.

Transform Interpolation

Although it is possible to find interpolated transforms and the Jacobian of interpolated transforms in constant time irrespective of how many states there are in the window, this would still add a significant amount of computation if it was done using equation 3.26 for every single feature point. To speed up computation, a discrete set of interpolated transforms and their Jacobians is precomputed at a finer resolution than the states themselves, and then a simpler interpolation scheme is used to approximate transforms & Jacobians when calculating measurement error.

Given a time t_i for which the interpolated transform and its Jacobian are required, the interpolation variable α is defined in Equation 3.35, where t_k and t_{k+1} are the times of the nearest precomputed transform such that $t_k \leq t_i \leq t_{k+1}$.

$$\alpha = \frac{t_i - t_k}{t_{k+1} - t_k} \tag{3.35}$$

Linear interpolation is used between the precomputed Jacobians while the transforms are linearly interpolated in se(3) according to Equation 3.36, where T_k, T_{k+1} are transforms for times t_k and t_{k+1}. The differences between the transforms ξ are also precomputed before evaluating measurement error.

$$\begin{aligned} T_k, T_{k+1} &\in \mathrm{SE}(3) \\ \xi &= T_{k+1} \boxminus T_k \\ T_\alpha &= T_k \boxplus (\alpha\xi) \end{aligned} \tag{3.36}$$

Chapter 4

Evaluation

4.1 Datasets

The proposed method is evaluated on the Kitti odometry dataset as well as on a dataset collected from a vehicle testbed in Waterloo, Ontario.

4.1.1 Kitti dataset

The Kitti dataset has emerged as a standard benchmark for visual and lidar odometry algorithms. The dataset consists of 22 sequences ranging from 1-9 minutes in length, and covering 39.2km. It includes residential, light urban, and highway driving trajectories collected in Karlsruhe, Germany. The lidar scans are provided pre-corrected for ego-motion. The lidar sensor used in this dataset is the Velodyne HDL-64e.

The 22 sequences are split into training and test sets, each having 11 sequences. The training sequences are provided with ground truth for developing algorithms, while the test sequences are provided without ground truth and are evaluated on a benchmark server hosted by the Karlsruhe Institute of Technology. In this thesis only the training portion of the dataset is evaluated.

4.1.2 Waterloo dataset

This dataset consists of a single sequence containing about half an hour of driving through 14.1km of commercial and residential areas of Waterloo, Ontario. Compared to the Kitti

dataset, the lidar scans are not corrected for ego-motion, and instead of a Velodyne HDL-64e, a Velodyne VLP-32C was used. This lidar is different in that is has half the number of laser-detector pairs spread over the same field of view. The elevation resolution is varied to concentrate most of the data around 0 degrees elevation with respect to the sensor. This is visible in Figure 2.1.

This differences of this dataset from the Kitti dataset provide more variety to check the consistency of any trends observed from results on the Kitti dataset. The lidar scans are uncorrected for ego-motion, from a different sensor, and collected in a North American instead of European scene.

The Waterloo dataset contains camera data from 8 roof-mounted cameras. Although images are not used for lidar odometry, a sample set of images taken in the same location as Figure 2.1 is shown in Figure 4.1 to provide a qualitative sense of the dataset. It can be seen that the different colours on the wall and the windows of the building also appear in the intensity data of the lidar, but that the lane markings are difficult to observe. This is because the lidar ground resolution is quite sparse and the lane markings themselves are mostly broken lines. This is why it is important to consider multiple scans when fitting intensity edges.

4.2 Configuration

The odometry is evaluated with a baseline configuration using only geometric edges against a configuration that adds intensity edges, and a configuration that increases the size of the sliding window.

The configurations are kept the same for both the Kitti and Waterloo dataset with the exception of the threshold to pick intensity edges. A different lidar sensor was used for each dataset, and the sensors have different intensity measurement characteristics. For the motion model, Q in Eq. 2.34 was found by starting from identity and hand-tuning.

4.2.1 Error Metric

The platforms used to collect each dataset are each equipped with a GPS-INS state estimation system. These systems fuse IMU measurements with real-time kinematic GPS to record the trajectory of the IMU in geocentric coordinates. The GPS-INS system provides a normally distributed error model as standard deviations on each coordinate. The latitude

(a) Front View (b) Rear View

(c) Right Front View (d) Right Rear View

Figure 4.1: Image Data Frame from Waterloo Dataset

and longitude standard deviation is on the order of a few centimeters for the majority of each sequence. The transform between the lidar and IMU is found using scan registration to build a consistent lidar map, and performing a hand-eye calibration [13] between the trajectory of the lidar and the IMU. The IMU to lidar transform is used to convert the IMU trajectory into a ground truth lidar trajectory. The output of lidar odometry is evaluated against the ground truth trajectory.

The proposed method is evaluated using the metric used by Kitti [20]. Each sequence is broken into as many subsequences of specific lengths as possible for that sequence and the error for each subsequence is evaluated. The error across all subsequences is averaged to provide a final statistic. All subsequences having lengths of 100, 200, 300, 400, 500, 600, 700, and 800 meters are considered.

Let $\mathcal{F}$ be the set of start and end indices for all the subsequences. The translation error is

$$E_{trans}(\mathcal{F}) = \frac{1}{|\mathcal{F}|} \sum_{(i,j)\in\mathcal{F}} \|((_s\hat{t}_{sj} - {}_s\hat{t}_{si}) - ({}_st_{sj} - {}_st_{si}))\|_2 \tag{4.1}$$

where ${}_s\hat{t}_{sk}$ is the estimated translation from the start of the sequence, ${}_st_{sk}$ is the ground truth translation from the start of the sequence, and $|\mathcal{F}|$ is the sum of the distance covered by each subsequence. Frame s is the sensor frame at the very beginning of the sequence.

Let $\mathcal{F}$ be the set of start and end indexes for all the subsequences. The rotation error is

$$E_{rot}(\mathcal{F}) = \frac{1}{|\mathcal{F}|} \sum_{(i,j)\in\mathcal{F}} \angle[((\hat{R}_{sj} \ominus \hat{R}_{si}) \ominus (R_{sj} \ominus R_{si}))] \tag{4.2}$$

where $\hat{R}_{sk}$ is the estimated rotation from the start of the sequence, and R_{sk} is the ground truth rotation from the start of the sequence, and the $\ominus$ operator is

$$R_i \ominus R_j = R_i^{-1} R_j \tag{4.3}$$

The $\ominus$ operator is distinct from the $\boxminus$ operator already introduced, and is only used within the evaluation metric.

The $\angle$ operator finds the magnitude of the rotation from a rotation matrix. It is defined by

$$\theta(R) = \arccos \frac{\mathrm{Tr}(R) - 1}{2} \tag{4.4}$$

where Tr is the matrix trace operator, and θ is the rotation magnitude.

4.3 Results

4.3.1 Kitti Results

Table 4.1: Average error on Kitti training sequences for sliding window size = 5. R.E. is rotational error and is in units of degrees / kilometer. T.E. is translation error and is expressed as a percentage of distance traveled. Geo indicates geometric features only, geo + int indicates geometric and intensity features. The lowest error for each sequence is bolded.

Sequence	Geo		Geo + int	
	R.E.	T.E.	R.E.	T.E.
0	7.45	1.55	**7.39**	**1.53**
1	**7.63**	**2.61**	8.45	2.75
2	**6.28**	1.54	6.74	**1.53**
3	8.04	1.15	**7.99**	**1.01**
4	7.41	1.17	**7.25**	**1.15**
5	5.23	.927	**5.11**	**.901**
6	4.86	.850	**4.55**	**.848**
7	**7.68**	1.05	8.31	**.981**
8	7.26	1.43	**6.93**	**1.42**
9	**6.29**	**1.46**	6.85	1.60
10	9.56	2.38	**9.42**	**2.13**

Table 4.2: Average error on Kitti training sequences for sliding window size = 10. R.E. is rotational error and is in units of degrees / kilometer. T.E. is translation error and is expressed as a percentage of distance traveled. Geo indicates geometric features only, geo + int indicates geometric and intensity features. The lowest error for each sequence is bolded.

Sequence	Geo		Geo + Int	
	R.E.	T.E.	R.E.	T.E.
0	**7.12**	1.47	7.24	**1.45**
1	**6.63**	**1.96**	8.23	2.46
2	6.26	1.57	**6.11**	**1.48**
3	7.95	1.22	**7.61**	**1.03**
4	**7.06**	1.11	7.55	1.11
5	**5.49**	**.965**	5.55	1.10
6	4.80	.923	**4.69**	**.883**
7	7.34	.937	**6.50**	**.712**
8	7.46	**1.39**	**7.29**	1.53
9	7.25	1.59	**6.71**	**1.44**
10	**8.72**	2.26	8.84	**1.98**

As shown in Table 4.1, the addition of intensity features provides a modest performance improvement on most of the sequences. Overall, sequences 1 and 10 perform significantly worse than the remaining sequences. Sequence 1 is a highway sequence, having higher average velocity than the rest of the dataset. Sequence 10 is relatively unstructured compared to the rest of the dataset, containing fewer buildings and more trees.

Table 4.2 shows the effect of adding intensity features when using a larger sliding window size. While the addition of intensity features improves the statistics of most of the sequences, the set of sequences that saw improvement is smaller than that using a smaller sliding window. Also, the preferred landmark configuration of a few sequences changed with the sliding window size.

The trajectory estimate for sequence 1, which resulted in the worst performance, is plotted in Figure 4.2 against the ground truth. The estimate for sequence 7, the best performing sequence is plotted in Figure 4.3.

Adding intensity edges produces a modest improvement in most sequences, but not all. It was expected that adding intensity edges would not improve performance across all sequences because there may not be useful intensity feature points in all sequences.

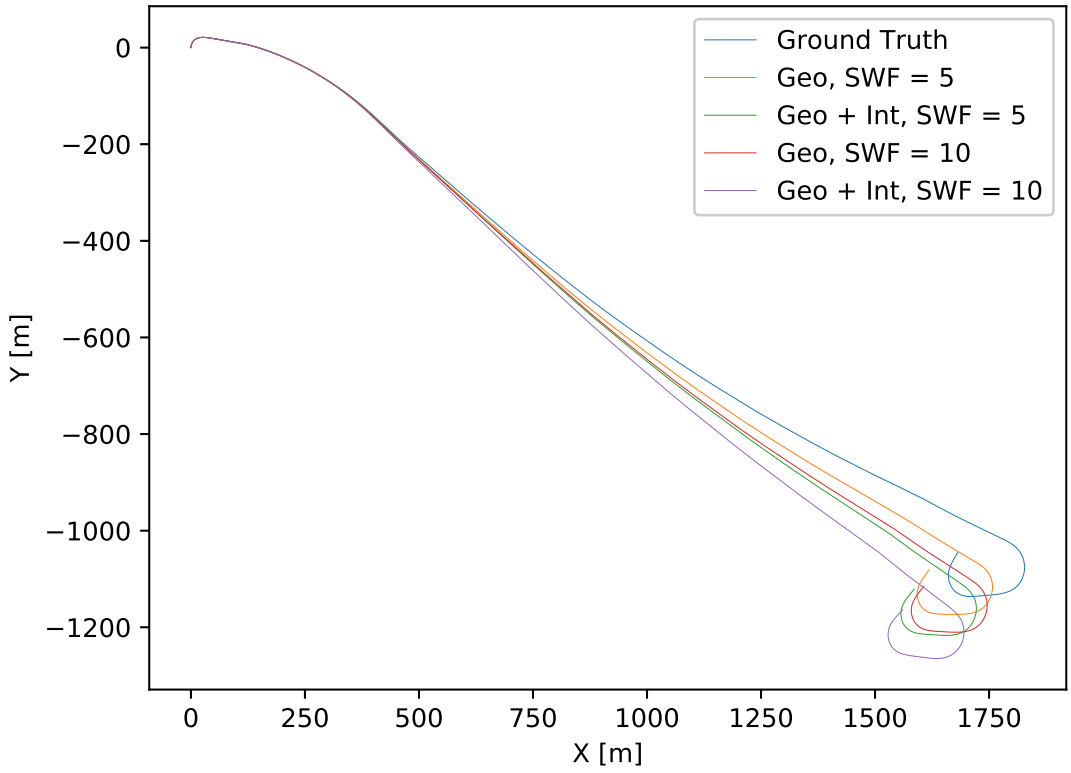

Figure 4.2: Odometry trajectory on Kitti sequence 1

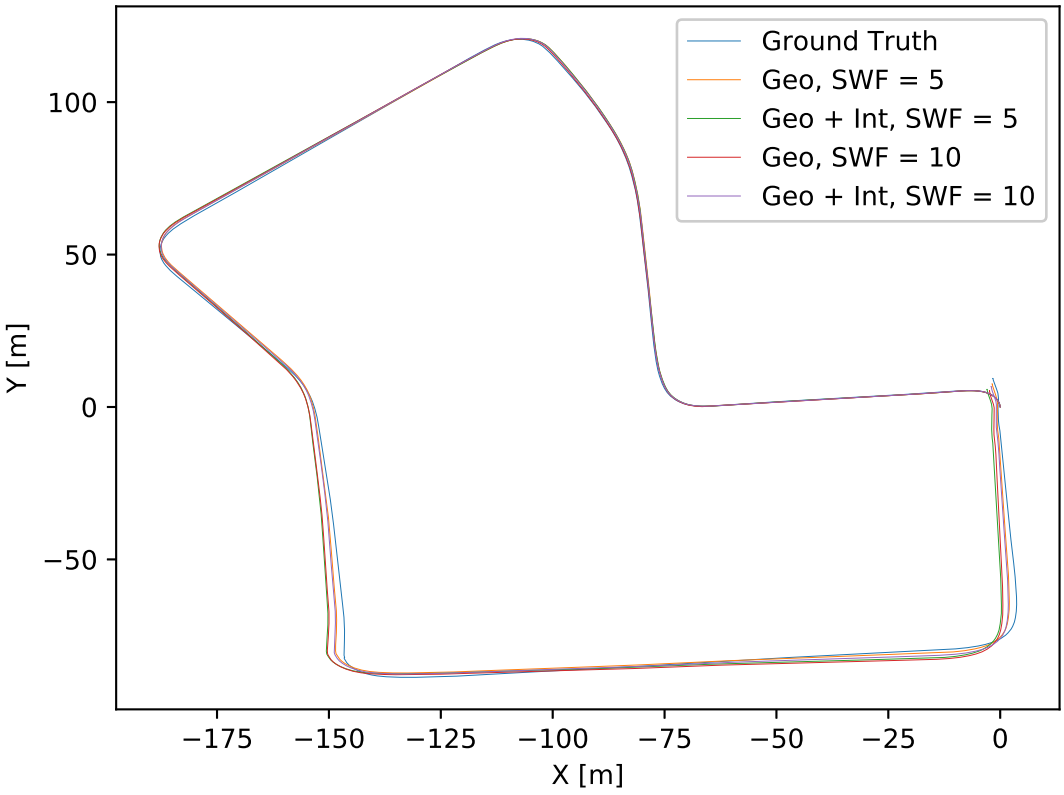

Figure 4.3: Odometry trajectory on Kitti sequence 7

However, the drop in performance for some sequences suggests that there is significant number of incorrectly corresponded feature points or that intensity edges are being fit to parts of the scene that do not meet the linear assumption.

Doubling the sliding window size from 5 to 10 resulted in better performance on some sequences, but not all. This may be due to the constant velocity motion model: the motion model has an averaging effect on the velocity states within the window, the strength of which depends on the strength of the constraint. The averaging effect is limited to the current window because marginalization is not used. Consequently, smaller window sizes can reduce the averaging effect of the motion model and may better track accelerations.

4.3.2 Waterloo Results

Table 4.3: Average error on Waterloo sequence. R.E. is rotational error and is in units of degrees / kilometer. T.E. is translation error and is expressed as a percentage of distance traveled. SWF refers to size of sliding window, geo denotes geometric features only, and geo + int denotes geometric and intensity features. The lowest error for each size of sliding window is bolded.

SWF = 3				SWF = 6			
Geo		Geo + Int		Geo		Geo + Int	
R.E.	T.E.	R.E.	T.E.	R.E.	T.E.	R.E.	T.E.
3.44	1.22	**3.27**	**1.19**	3.09	1.12	3.09	**1.10**

The results on the Waterloo sequence are shown in Table 4.3. The addition of intensity edges results in a modest improvement in average statistics on this sequence. The relatively small improvement might be because where this sequence contains useful lane markings, there are already well-defined curbs which are modeled well by geometric edge landmarks. Increasing the size of the sliding window improves performance both in the case of geometric only features and with the addition of intensity features.

The results of this sequence are plotted against ground truth in Figure 4.4. It shows that despite the close average statistics, there is a difference between configurations visible due to the long sequence length.

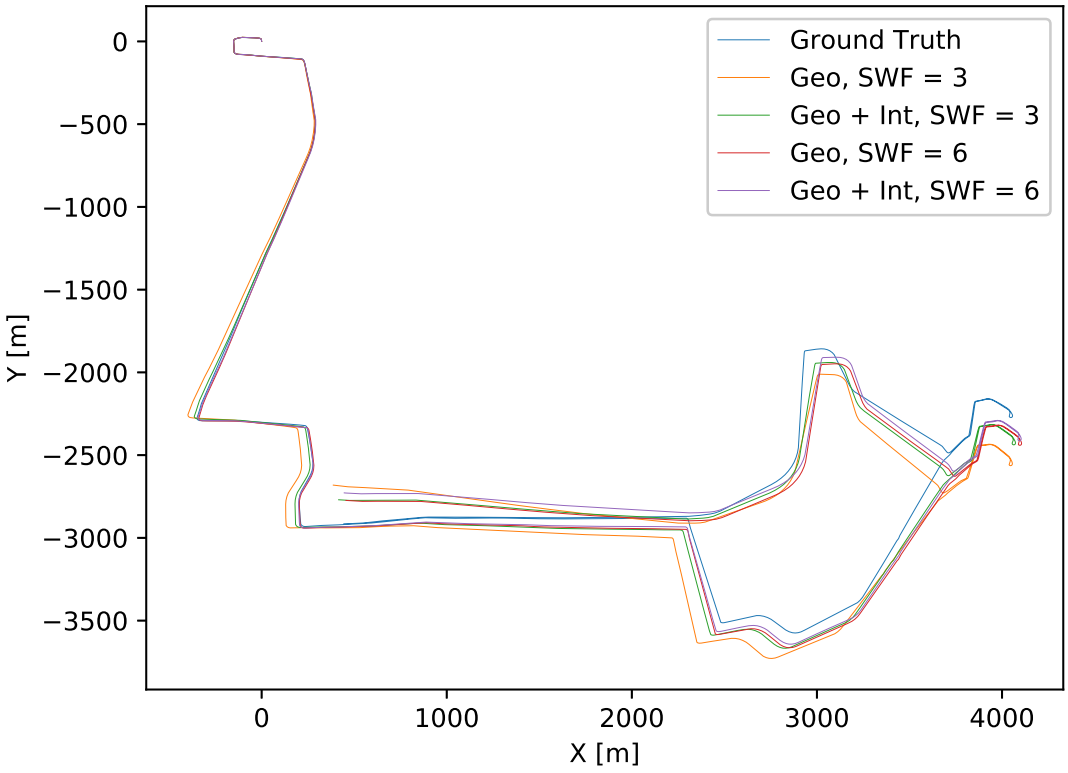

Figure 4.4: Trajectory plots for the different configurations specified in Table 4.3 for the Waterloo sequence

4.4 Discussion

The results shown indicate that the intensity channel on these datasets can be used in conjunction with geometric information for an improvement in performance. The sequences that show a loss of performance indicate that the current model for intensity landmarks could be being applied incorrectly in some cases. A possible solution would be to redesign how intensity landmarks are initialized and/or explore whether model other than a line is more suitable for modeling scene elements.

The performance of the presented lidar odometry on Kitti does not achieve that of the current state-of-the-art lidar odometry methods. However, there are some likely explanations with associated solutions to improve performance, and the performance that was achieved is likely sufficient for this algorithm to be used as the relative pose estimation within a larger SLAM implementation that includes loop closure.

The relatively poor performance on sequence 1 in the Kitti dataset indicates that speed is likely a contributing factor to estimation error for this algorithm. This might be due to the constant velocity motion model. The route includes getting on and off a highway, which involves large sustained acceleration. Landmarks in the scene are also sampled less at high speed because the lidar is not able to make as many observations while passing by.

4.5 Possible Improvements

Landmark Initialization

The landmarks are currently initialized with a fairly simple approach, relying on the subsequent robust loss to remove the influence of bad landmarks. The edge technique in particular is prone to initializing edge landmarks in the midst of noisy clouds of points, such as that from foliage, because the edge initialization merely attempts to find coincident points. Outlier rejection methods like RANSAC could be applied to clusters of points to initialize new landmarks more robustly.

Only the current and previous sets of feature points are considered when initializing new features. This is sometimes problematic because two scans may not contain enough samples to reliably initialize landmarks, particularly lane markings when using the VLP-32C in the Waterloo dataset, whose nonlinear beam spacing results in coarse ground resolution. Using a longer history of feature points would provide richer information to initialize edges.

Finally, the edge landmark initialization algorithm is computationally expensive due to the evaluation of all the possible line samples. Finding a more efficient algorithm would reduce the runtime.

Edge Landmark Model

The edge landmarks are well suited to model true edges, like corners on buildings or curbs. However, they are also used to model poles and tree trunks in the environment. Since these objects are approximately cylindrical, the position of the edge as seen by the lidar changes as the sensor moves past the object. This is a source of systematic error because there is an implicit assumption that edges in the scene are static. This could be resolved by either developing a filter to remove edge features sampled from poles, or by introducing another landmark type that is able to model these objects more accurately.

Sampling Strategy

Increasing the number of landmarks and feature points is a trade-off between using as much information as possible and computational effort. However, not all landmarks and feature points are equal: if there is a landmark that constrains yaw, then the additional information gained from adding another landmark that constrains yaw is diminished. A sampling strategy seeks to maximize the information gained given an allowance of landmarks by controlling how they are placed.

Scan sampling is used during feature point extraction and new landmark initialization. The technique to spread feature points and landmarks over polar bins places these elements approximately evenly throughout the scene. However, better sampling strategies such as geometrically stable sampling [21] exist and could be used to improve the distribution of landmarks in the scene.

Sliding Window Updates

A problem with the current approach is that during periods when the car is stopped, eventually all the scans in the sliding window are taken from the same physical location. This has an adverse effect on landmark initialization and optimization because the benefit of having landmarks sampled by multiple scans from multiple positions is lost. Further, the feature points from each duplicate scan all create measurements, so when the car starts moving again, the measurements from the stopped position are over-weighted relative to

the new scans. Introducing a distance-based sliding window update similar to the idea of keyframes would solve this problem.

Chapter 5

Conclusion

Odometry is a fundamental component in all state-of-the-art SLAM systems. Lidar is likely to be an important component of SLAM systems operating on self-driving cars because unlike cameras, lidar offers consistent performance regardless of ambient lighting conditions. This is a large advantage when developing systems that must have a guarantee of safety as with self-driving cars, and there is enough competition amongst lidar manufacturers that lidar sensors will not be prohibitively expensive for this application.

This thesis presents a lidar odometry algorithm that formulates the lidar odometry problem using explicit landmark states, and is solved within a sliding window filter. The algorithm takes into consideration the anisotropic resolution characteristic of the lidar sensors commonly found on automotive platforms, and leverages the intensity returns measured by lidar as well as the geometric measurements.

The lidar odometry algorithm is evaluated on the Kitti odometry training dataset as well as on a private dataset collected within the City of Waterloo. It achieves between 4.6 and 9.6 degrees per kilometer of average rotational drift and between 0.71% and 2.8% average translation drift on the Kitti dataset depending on the sequence and configuration. It achieves between 3.1 and 3.4 degrees per kilometer of average rotation drift and between 1.1% and 1.2% average translation drift on the Waterloo dataset. The addition of intensity information is shown to improve performance on most of the Kitti sequences as well as on the Waterloo dataset. Increasing the size of the sliding window is shown to improve performance on most sequences, including the Waterloo dataset.

Safety-critical odometry applications such as self-driving cars must be robust as well as accurate. The work in this thesis uses both geometric and intensity features in an attempt to capitalize on as much of the sensor's information as possible, to maintain performance

in areas lacking geometric features, while also using a lightweight scene model. Continuing work on the algorithm introduced could lead to more robust, standalone lidar odometry.

5.1 Future Work

Motion Model

The motion model weight significantly affects the performance of the odometry because it controls the trade-off between reducing noisy estimation error and total bandwidth of the estimate. In this work, the motion model weights were hand-tuned until the trajectories produced were qualitatively good. Formulating and solving the problem of finding the optimal weight given the datasets as an optimization would result in a weight that would yield better results.

The motion model used is based on a constant velocity assumption. Clearly, the assumption is violated frequently while driving, so the motion model must be adding some systematic error. The STEAM formulation has recently been demonstrated with a constant acceleration motion model and was found to improve performance compared to a constant velocity motion model [35]. Even though a constant acceleration assumption is also violated by a driving car in some circumstances, it can model constant braking, constant throttle, and the transition from straight line motion to turning.

Formulating a motion model based on the specific dynamics of the platform (car) as a GP and using it within the STEAM framework would reduce systematic error caused by the motion model currently used.

Incorporating throttle, brake, and steering input into the motion model would further reduce systematic error.

Mapping

The application of the scene model to mapping and online localization is not explored in this work, but the scene model is well-suited for fast, online localization due to the sparsity of the landmarks compared to a dense pointcloud map. A map of landmarks would require far less storage than a high resolution, dense pointcloud map since a relatively small amount of landmarks can constrain the trajectory.

As the landmark states are continuous, a map of landmarks avoids the spatial discretization of voxel representations. This is advantageous because a discrete map must

have a sufficiently fine resolution for localization. Even with efficient data representations like octrees [33], finer resolution increases the storage requirements. This trade-off is not present with the proposed scene model, instead, the storage requirement is dependent on the number of landmarks, and if the scene contains a surplus of landmarks, some may be omitted from the map.

Camera Integration

Cameras have finer resolution than the lidars used on automotive platforms today, and state-of-the-art visual odometry is of similar quality to that of lidar odometry. A logical approach is to combine cameras and lidar into a single odometry algorithm. This idea has been explored by other methods, one of which [39] is currently the top performing method on the Kitti odometry test evaluation.

Given that both cameras and lidars can detect lane markings, and that geometric edges can be visually distinct, it is likely that some of the landmarks currently used could be measured by camera as well as by lidar. Formulating camera & lidar measurements as functions of the same state would offer a tightly-coupled approach. The extra detail captured by a camera may also aid the initialization of good landmarks.

References

[1] Full STEAM ahead: Exactly sparse Gaussian process regression for batch continuous-time trajectory estimation on SE(3). *IEEE International Conference on Intelligent Robots and Systems (IROS), 2015*, 2015-Decem(3):157–164, 2015.

[2] Evan Ackerman and Erico Guizzo. irobot brings visual mapping and navigation to the roomba 980, 2015.

[3] Sameer Agarwal, Keir Mierle, and Others. Ceres solver. http://ceres-solver.org.

[4] Tim Barfoot, Chi Hay Tong, and Simo Sarkka. Batch Continuous-Time Trajectory Estimation as Exactly Sparse Gaussian Process Regression. 2014.

[5] Timothy D Barfoot. *State Estimation for Robotics: A Matrix-Lie-Group Approach*. Cambridge University Press, 2016.

[6] Paul J. Besl and Neil D. McKay. A Method for Registration of 3-D Shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, 1992.

[7] Peter Biber and W. Strasser. The normal distributions transform: a new approach to laser scan matching. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2003*, volume 3, pages 2743–2748, 2003.

[8] M. Bosse and R. Zlot. Continuous 3d scan-matching with a spinning 2d laser. In *IEEE International Conference on Robotics and Automation (ICRA), 2009*, pages 4312–4319, 2009.

[9] Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, Jose Neira, Ian Reid, and John J. Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on Robotics*, 32(6):1309–1332, 2016.

[10] L. E. Clement, V. Peretroukhin, J. Lambert, and J. Kelly. The battle for filter supremacy: A comparative study of the multi-state constraint kalman filter and the sliding window filter. In *2015 12th Conference on Computer and Robot Vision*, pages 23–30, 2015.

[11] Gabriele Costante, Michele Mancini, Paolo Valigi, and Thomas A. Ciarfuglia. Exploring Representation Learning With CNNs for Frame-to-Frame Ego-Motion Estimation. *IEEE Robotics and Automation Letters*, 1(1):18–25, 2016.

[12] Jean-Emmanuel Deschaud. IMLS-SLAM: scan-to-model matching based on 3D data. pages 1–6.

[13] F. Dornaika and R. Horaud. Simultaneous robot-world and hand-eye calibration. *IEEE Transactions on Robotics and Automation*, 14(4):617–622, 1998.

[14] D. Droeschel and S. Behnke. Efficient continuous-time slam for 3d lidar-based online mapping. In *IEEE International Conference on Robotics and Automation (ICRA), 2018*, pages 1–9, 2018.

[15] Daniel Fagnant and Kara Kockelman. Preparing a nation for autonomous vehicles: Opportunities, barriers and policy recommendations, October 2013.

[16] O P Ferreira, A N Iusem, and S. Z. Németh. Concepts and techniques of optimization on the sphere. *TOP*, 22(3):1148–1170, 2014.

[17] Christian Forster, Luca Carlone, Frank Dellaert, and Davide Scaramuzza. Imu preintegration on manifold for efficient visual-inertial maximum-a-posteriori estimation. In *Robotics: Science and Systems (RSS)*, 2015.

[18] Paul Furgale. Representing robot pose: The good, the bad, and the ugly, June 2014.

[19] Paul Furgale, T D Barfoot, and G Sibley. Continuous-time batch estimation using temporal basis functions. In *IEEE International Conference on Robotics and Automation (ICRA), 2012*, pages 2088–2095, 2012.

[20] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3354–3361. IEEE, 2012.

[21] N. Gelfand, L. Ikemoto, S. Rusinkiewicz, and M. Levoy. Geometrically stable sampling for the icp algorithm. In *Fourth International Conference on 3-D Digital Imaging and Modeling (3DIM), 2003*, pages 260–267, 2003.

[22] Adam Harmat, Michael Trentini, and Inna Sharf. Multi-camera tracking and mapping for unmanned aerial vehicles in unstructured environments. *Journal of Intelligent & Robotic Systems*, 78(2):291–317, May 2015.

[23] Andrew Johnson. *Spin-Images: A Representation for 3-D Surface Matching*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, August 1997.

[24] Ralf Kaestner, Sebastian Thrun, Michael Montemerlo, and Matt Whalley. A Non-Rigid Approach to Scan Alignment and Change Detection Using Range Sensor Data. *Field and Service Robotics: Results of the 5th International Conference STAR*, pages 1–12, 2006.

[25] Anthony W. Knapp. *Lie groups beyond an introduction*. Birkhauser, 1996.

[26] J. Levinson and S. Thrun. Robust vehicle localization in urban environments using probabilistic maps. In *IEEE International Conference on Robotics and Automation (ICRA), 2010*, pages 4372–4378, 2010.

[27] Montiel J. M. M. Mur-Artal, Raúl and Juan D. Tardós. ORB-SLAM: a versatile and accurate monocular SLAM system. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015.

[28] Austin Nicolai, Ryan Skeele, Christopher Eriksen, and Geoffrey A Hollinger. Deep learning for laser based odometry estimation. In *Robotics: Science and Systems Conference Workshop on Limits and Potentials of Deep Learning in Robotics (RSS)*, 2016.

[29] Edward Rosten and Tom Drummond. Machine Learning for High Speed Corner Detection. *Computer Vision – ECCV 2006*, 1:430–443, 2006.

[30] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. ORB: An efficient alternative to SIFT or SURF. *Proceedings of the IEEE International Conference on Computer Vision*, pages 2564–2571, 2011.

[31] Aleksandr V Segal, Dirk Haehnel, and Sebastian Thrun. Generalized-ICP. *Robotics: Science and Systems (RSS)*, 2:4, 2009.

[32] J. Servos and S. L. Waslander. Multi channel generalized-icp. In *IEEE International Conference on Robotics and Automation (ICRA), 2014*, pages 3644–3649, 2014.

[33] F. Steinbrcker, J. Sturm, and D. Cremers. Volumetric 3d mapping in real-time on a cpu. In *IEEE International Conference on Robotics and Automation (ICRA), 2014*, pages 2021–2028, 2014.

[34] G. K. L. Tam, Z. Cheng, Y. Lai, F. C. Langbein, Y. Liu, D. Marshall, R. R. Martin, X. Sun, and P. L. Rosin. Registration of 3d point clouds and meshes: A survey from rigid to nonrigid. *IEEE Transactions on Visualization and Computer Graphics*, 19(7):1199–1217, 2013.

[35] Tim Y. Tang, David J. Yoon, and Timothy D. Barfoot. A white-noise-on-jerk motion prior for continuous-time trajectory estimation on SE(3). *CoRR*, abs/1809.06518, 2018.

[36] Chi Hay Tong, Sean Anderson, Hang Dong, and Timothy D. Barfoot. Pose interpolation for laser-based visual odometry. *Journal of Field Robotics*, 31(5):787–813, 2014.

[37] Sen Wang, Ronald Clark, Hongkai Wen, and Niki Trigoni. DeepVO: Towards end-to-end visual odometry with deep Recurrent Convolutional Neural Networks. In *IEEE International Conference on Robotics and Automation (ICRA), 2017*, pages 2043–2050. IEEE, 2017.

[38] Georges Younes, Daniel Asmar, Elie Shammas, and John Zelek. Keyframe-based monocular SLAM: design, survey, and future directions. *Robotics and Autonomous Systems*, 98:67–88, 2017.

[39] J. Zhang and S. Singh. Visual-lidar odometry and mapping: low-drift, robust, and fast. In *IEEE International Conference on Robotics and Automation (ICRA), 2015*, pages 2174–2181, 2015.

[40] Ji Zhang and Sanjiv Singh. Loam: Lidar odometry and mapping in real-time. volume 32, pages 141–148, 2015.

APPENDICES

Appendix A: Jacobian of Interpolated Transforms

The Jacobian of the interpolated transforms with respect to the state parameters is required. Beginning with estimates for the state at discrete times t_k and t_{k+1}, the interpolated state between those times is defined by using Eq. 2.28 [4], the specific transition matrix from Eq. 2.30, then converting the local variables to global variables.

The local variables are defined by Equation 1, where J is the Jacobian of the SE3 exponential map and time $t_{k+1} \geq t_k$. The local variables are with respect to the state at time t_k, and the time of the desired interpolated transform is somewhere between times t_k, t_{k+1}.

$$\gamma_k(t_k) = \begin{bmatrix} 0 \\ \varpi(t_k) \end{bmatrix} \gamma_k(t_{k+1}) = \begin{bmatrix} T_{k+1} \boxminus T_k \\ J(T_{k+1} \boxminus T_k)^{-1} \varpi_{k+1} \end{bmatrix} \tag{1}$$

Equation 2 is the interpolated state in global variables [5]. Interpolation variables Ψ and Λ are dependent on the motion model, and are defined in Equation. 2.28.

$$\begin{aligned} T(t) &= \exp((\Lambda_1(t)\gamma_k(t_k) + \Psi_1(t)\gamma_k(t_{k+1})^\wedge))T_k \\ \varpi(t) &= J(T(t) \boxminus T_k)(\Lambda_2(t)\gamma_k(t_k) + \Psi_2(t)\gamma_k(t_{k+1})) \end{aligned} \tag{2}$$

where

$$\gamma_k(t) = \Lambda(t)\gamma_k(t_k) + \Psi(t)\gamma_k(t_{k+1})$$

$$\Lambda(t) = \begin{bmatrix} \Lambda_1(t) \\ \Lambda_2(t) \end{bmatrix}$$
$$\Psi(t) = \begin{bmatrix} \Psi_1(t) \\ \Psi_2(t) \end{bmatrix}$$

In this application, corresponding points must be expressed in the same frame to evaluate residuals. Let the time of a point be $t_A \in [t_k, t_{k+1}], t_k < t_{k+1}$. The point will be transformed to the sensor frame at time t_k. The pose at times t_k, t_{k+1} are states. One of the properties of SE(3) is tangent perturbations applied on the right can be converted to tangent perturbations applied on the left. This is done with the adjoint operator $\curlywedge$. This property is defined in Eq. 3.

$$\begin{aligned} \exp((T^{\curlywedge}\xi)^{\wedge})T &= T\exp(\xi^{\wedge}) \\ T &= \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix}, T \in \mathrm{SE}(3) \\ T^{\curlywedge} &= \begin{bmatrix} R & 0 \\ t^{\wedge}R & R \end{bmatrix} \end{aligned} \tag{3}$$

The difference between two elements of SE(3) is written as in Equation 4.

$${}_k\xi_{k+1} = T_{k+1} \boxminus T_k \tag{4}$$

Equation 5 a linear approximation for the difference between two elements of SE(3) both subject to perturbations. It is linearized with respect to the perturbations when the perturbations are zero. It is therefore a reasonable approximation for small perturbations.

$$\begin{aligned} (T_{k+1} \boxplus \epsilon_{k+1}) \boxminus (T_k \boxplus \epsilon_k) &= \ln(\exp(\epsilon_{k+1})T_{k+1}(\exp(\epsilon_k)T_k)^{-1})^{\vee} \\ &= \ln(\exp(\epsilon_{k+1})T_{k+1}T_k^{-1}\exp(-\epsilon_k))^{\vee} \\ &= \ln(\exp(\epsilon_{k+1})\exp(-(T_{k+1}T_k^{-1})^{\curlywedge}\epsilon_k)T_{k+1}T_k^{-1})^{\vee} \\ &\approx \ln(\exp(\epsilon_{k+1} - (T_{k+1}T_k^{-1})^{\curlywedge}\epsilon_k)T_{k+1}T_k^{-1})^{\vee} \\ &\approx J({}_k\xi_{k+1})^{-1}(\epsilon_{k+1} - (T_{k+1}T_k^{-1})^{\curlywedge}\epsilon_k) + {}_k\xi_{k+1} \end{aligned} \tag{5}$$

To better show that Equation 5 is linear with respect to the perturbations, Equation 6 defines Θ.

$$\Theta = \begin{bmatrix} -J({}_k\xi_{k+1})^{-1}(T_{k+1}T_k^{-1})^{\curlywedge} & J({}_k\xi_{k+1})^{-1} \end{bmatrix} \tag{6}$$

With Θ, Equation 5 is equivalent to Equation 7.

$$J({}_k\xi_{k+1})^{-1}(\epsilon_{k+1} - (T_{k+1}T_k^{-1})^{\curlywedge}\epsilon_k) + {}_k\xi_{k+1} = \Theta \begin{bmatrix} \epsilon_k \\ \epsilon_{k+1} \end{bmatrix} + {}_k\xi_{k+1} \tag{7}$$

Substituting Equations 2.9 & 1 into Eq 2 leads to Eq 8, which is an expression for the interpolated transform given the states and perturbation on the states at times t_k, t_{k+1}. The variable λ used here as a shorthand for $(T_{k+1} \boxplus \epsilon_{k+1}) \boxminus (T_k \boxplus \epsilon_k)$.

$$T_A = \exp((\Lambda_1(t_A) \begin{bmatrix} 0 \\ \varpi_k + \psi_k \end{bmatrix} + \Psi_1(t_A) \begin{bmatrix} \lambda \\ J(\lambda)^{-1}(\varpi_{k+1} + \psi_{k+1}) \end{bmatrix})^{\wedge}) \exp(\epsilon_k) T_k \tag{8}$$

Using Equations 7 and 4 in Equation 8 and rearranging yields Eq. 9, which is the interpolated transform expression after incorporating the approximations of equation 5.

$$\begin{aligned} T_A \approx \exp((\Lambda_1(t_A) \begin{bmatrix} 0 \\ \psi_k \end{bmatrix} + \Psi_1(t_A) \begin{bmatrix} \Theta \begin{bmatrix} \epsilon_k \\ \epsilon_{k+1} \end{bmatrix} \\ J(\Theta \begin{bmatrix} \epsilon_k \\ \epsilon_{k+1} \end{bmatrix} + {}_k\xi_{k+1})^{-1}\psi_{k+1} \end{bmatrix} + \\ \Lambda_1(t_A) \begin{bmatrix} 0 \\ \varpi_k \end{bmatrix} + \Psi_1(t_A) \begin{bmatrix} {}_k\xi_{k+1} \\ J(\Theta \begin{bmatrix} \epsilon_k \\ \epsilon_{k+1} \end{bmatrix} + {}_k\xi_{k+1})^{-1}(\varpi_{k+1}) \end{bmatrix})^{\wedge}) \exp(\epsilon_k) T_k \end{aligned} \tag{9}$$

Equation 9 is first approximated by dropping the terms that are linear with respect to the transform perturbations that are in the arguments to the Jacobian of the exponential map. These terms are relatively small, and the approximation becomes exact when the perturbations are zero. Equation 10 defines that approximation.

$$
\begin{aligned}
T_A \approx \exp((&\Lambda_1(t_A)\begin{bmatrix}0\\ \psi_k\end{bmatrix} + \Psi_1(t_A)\begin{bmatrix}\Theta\begin{bmatrix}\epsilon_k\\ \epsilon_{k+1}\end{bmatrix}\\ J({}_k\xi_{k+1})^{-1}\psi_{k+1}\end{bmatrix} + \\
&\Lambda_1(t_A)\begin{bmatrix}0\\ \varpi_k\end{bmatrix} + \Psi_1(t_A)\begin{bmatrix}{}_k\xi_{k+1}\\ J({}_k\xi_{k+1})^{-1}(\varpi_{k+1})\end{bmatrix})^\wedge)\exp(\epsilon_k)T_k
\end{aligned}
\tag{10}
$$

Equation 11 introduces additional variables to further shorten the expression and to separate the perturbations from the non-perturbations within Eq 10.

$$
\begin{aligned}
\Upsilon &= \begin{bmatrix}\Theta & \Lambda_1(t_A) & \Psi_1(t_A)J({}_k\xi_{k+1})^{-1}\end{bmatrix}\\
\Xi &= \Lambda_1(t_A)\begin{bmatrix}0\\ \varpi_k\end{bmatrix} + \Psi_1(t_A)\begin{bmatrix}{}_k\xi_{k+1}\\ J({}_k\xi_{k+1})^{-1}(\varpi_{k+1})\end{bmatrix}\\
x &= \begin{bmatrix}\epsilon_k\\ \epsilon_{k+1}\\ \psi_k\\ \psi_{k+1}\end{bmatrix}
\end{aligned}
\tag{11}
$$

Using the additional variables introduced, Equation 10 is now compactly stated as Equation 12.

$$
T_A \approx \exp(\Upsilon x + \Xi)\exp(\epsilon_k)T_k \tag{12}
$$

Starting from Equation 12, the exponential map is linearized at Ξ to yield Equation 13.

$$
\begin{aligned}
T_A &\approx \exp(\Xi) \boxplus (J(\Xi)\Upsilon x)\exp(\epsilon_k)T_k\\
&= \exp(J(\Xi)\Upsilon x)\exp(\Xi)\exp(\epsilon_k)T_k
\end{aligned}
\tag{13}
$$

The adjoint property is used to move $\exp(\epsilon_k)$ to the other side of $\exp(\Xi)$ in Equation 14.

$$T_A \approx= \exp(J(\Xi)\Upsilon x)\exp(\exp(\Xi)^{\curlywedge}\epsilon_k)\exp(\Xi)T_k \tag{14}$$

The last approximation used in this derivation is a truncated Baker-Campbell-Hausdorff formula to approximate $\exp(J(\Xi)\Upsilon x)\exp(\exp(\Xi)^{\curlywedge}\epsilon_k)$ with $\exp(J(\Xi)\Upsilon x + \exp(\Xi)^{\curlywedge}\epsilon_k)$. This approximation has error proportional to the square of the arguments, but since the arguments are all on the scale of the perturbations, it is an acceptable approximation.

Equation 15 finishes the derivation.

$$\begin{aligned} T_A &\approx \exp(J(\Xi)\Upsilon x + \exp(\Xi)^{\curlywedge}\epsilon_k)\exp(\Xi)T_k \\ &= exp(\Upsilon' x)\bar{T}_A \\ \text{where} & \\ \Upsilon' &= J(\Xi)\Upsilon + \begin{bmatrix}\exp(\Xi)^{\curlywedge} & 0 & 0 & 0\end{bmatrix} \\ \bar{T}_A &= \exp(\Xi)T_k \end{aligned} \tag{15}$$

Variable Υ' depends on the states before and after the interpolation time, as well as the interpolation time itself. In the main body of the text, Equation 15 appears as Equation 3.26, except Υ' has been split into two interpolation factors A_1, A_2 and the dependence on the states and interpolation time is made explicit.